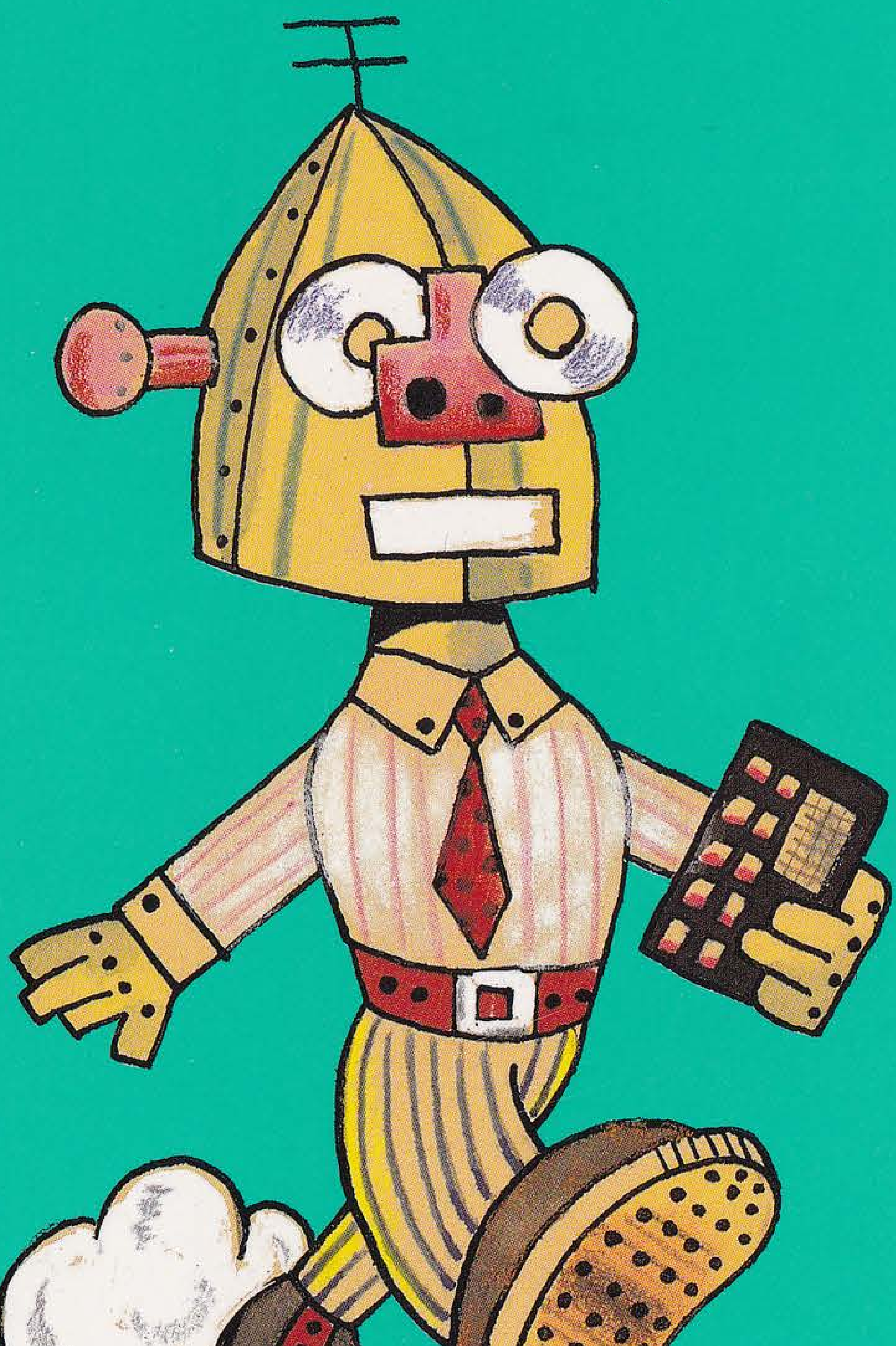
MSX POCKET BANK 3

ポケットバンク

BASICゲーム教室

安田吾郎



遊んで、作って、また遊ぶ
ゲーム作りのノウハウが
わかってしまう一冊

アスキー出版局

MSX POCKET BANK 3

BASICゲーム教室

安田吾郎

アスキー出版局

表紙デザイン　HIT STUDIO
表紙イラストレーション　加藤裕将
本文イラストレーション　藤岡　仁

はじめに
BASIC GAME

『自分もこんなにおもしろいゲームを作れるのか！』

そんな喜びをみなさんにも味わってほしいと思い，この本を書くことにしました．

マイコンでゲームを作るには，まずプログラムを作れることが必要です．本書を順を追って読んでいけば，だんだんとプログラムを作る力がついていきます．また，マイコンを買ったばかりでBASICはまだよくわからない，という人でも，書いてあることをひとつひとつ実行していけば十分理解できると思います．

要はBASICが動く機械を持ち，ゲームにたいする多少の興味があればよいのです．

ゲームを作るにはプログラムが作れなければなりません．しかし，ただプログラムを作れる，あるいはBASICの命令を知っているというだけで，おもしろいゲームが作れるというものでもありません．

たしかに簡単なものであれば，誰でも作れるのかもしれませんが，本当に楽しめて熱中できるゲームを作るには，それなりのノウハウが必要となるのです．

本書ではどうしたら他人も自分も楽しめるようなゲームが作れるか，そのコツをわかりやすく書いたつもりです．具体的には，以前ゲームセンターなどにも置いてあった「バリケードゲーム」や「ブロックくずし」を実際に作ってみます，プログラム作りには多少の自信があるという人にも参考になると思います．

なお本書のプログラムは，すべてMSX規格のBASICで書かれています．MSXマシンをお持ちの方は，本書に載っているプログラムを打ちこんでみてください．自分で打ちこむことにより理解はより深まるでしょうし，ただそのゲームで遊ぶだけでも楽しめると思います．

読者のみなさんの中から，将来ゲームプログラムのコンテストなどで優勝できるほどの人が現れれば，筆者にとって望外の喜びです．いや，やはり新たなライバルの出現として恐れるべきでしょうか？

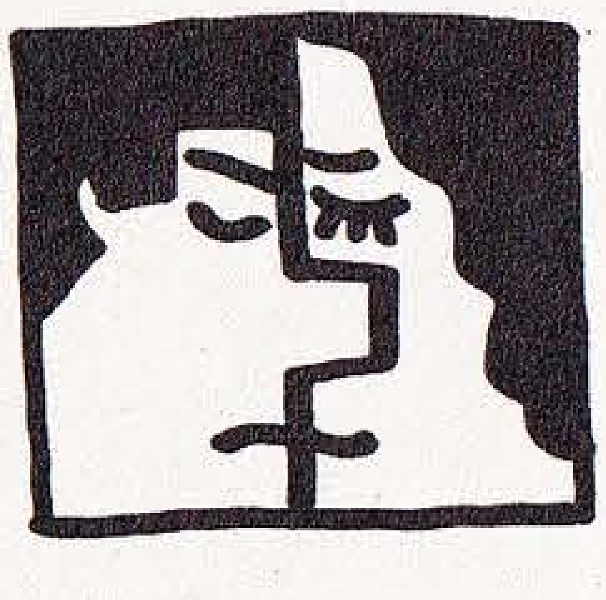
昭和59年3月3日　安田吾郎

CONT

ENTS

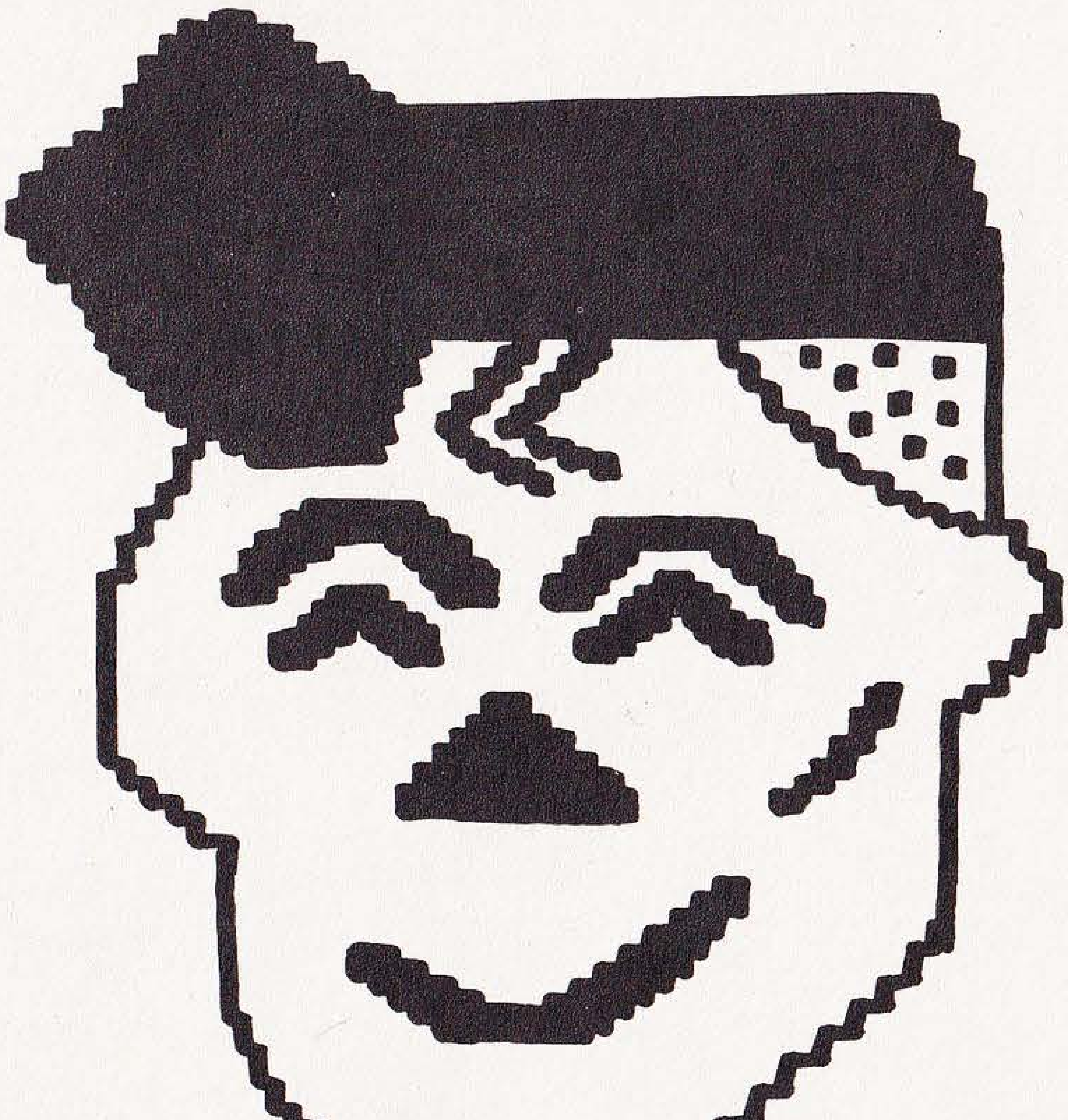
3 南極米・ソ戦ゲーム（実践編II）

4 ブロックくずし（実践編III）



1

ゲーム作りの基礎



いきなりゲーム作りにはいる前に，まずこの章ではそのための足場をかためることにしましょう．「急がば回れ」と言うように，本物の応用力をつけるためにはやはり基本的なところをおさえておく必要があるのです．

1節では，マイコンゲームにはどのような種類があるのかについてふれます．

2節では，ゲームを作るにはどのような手順を取ればよいかという点を解説します．これをマスターするのがいわば本書の主目的です．

3節では，ゲームを作るにはどの程度BASICがわかっていればよいのかについて述べてみます．付録の**重要度別BASICコマンド表**は新しい試みで，プログラマーをめざす入門者には，どこから勉強したらよいのかを知る指針となるでしょう．

そして2章以降は，1章で述べたことの実践として，3つのゲームを作っていくことにします．

1節 マイコンゲームあれこれ

マイコンゲームというと皆さんはどのようなゲームを思い浮かべるでしょうか？　ブロックくずし，インベーダーゲーム，パックマン，ジャンピューターといったあたりが，まず頭に浮かぶのではないでしょうか．これらのゲームは，一般にアーケードゲームと呼ばれ，インベーダーやパックマンのように反射神経を競うタイプ，ジャンピューターのように思考を要するタイプ，コンピュータ花札やブラックジャックのように運の要素が強いタイプ，の3つにわけることができます．

しかし，実際のマイコンゲームにはアーケードゲームにない他のものがあります．ゲームの終了までに1ヵ月以上も要するアドベンチャーゲームなどです．

それではこういったゲームも含めて，マイコンゲームをいくつかに分類してみましょう．

1．アクションゲーム(インベーダーゲーム，パックマンなど)

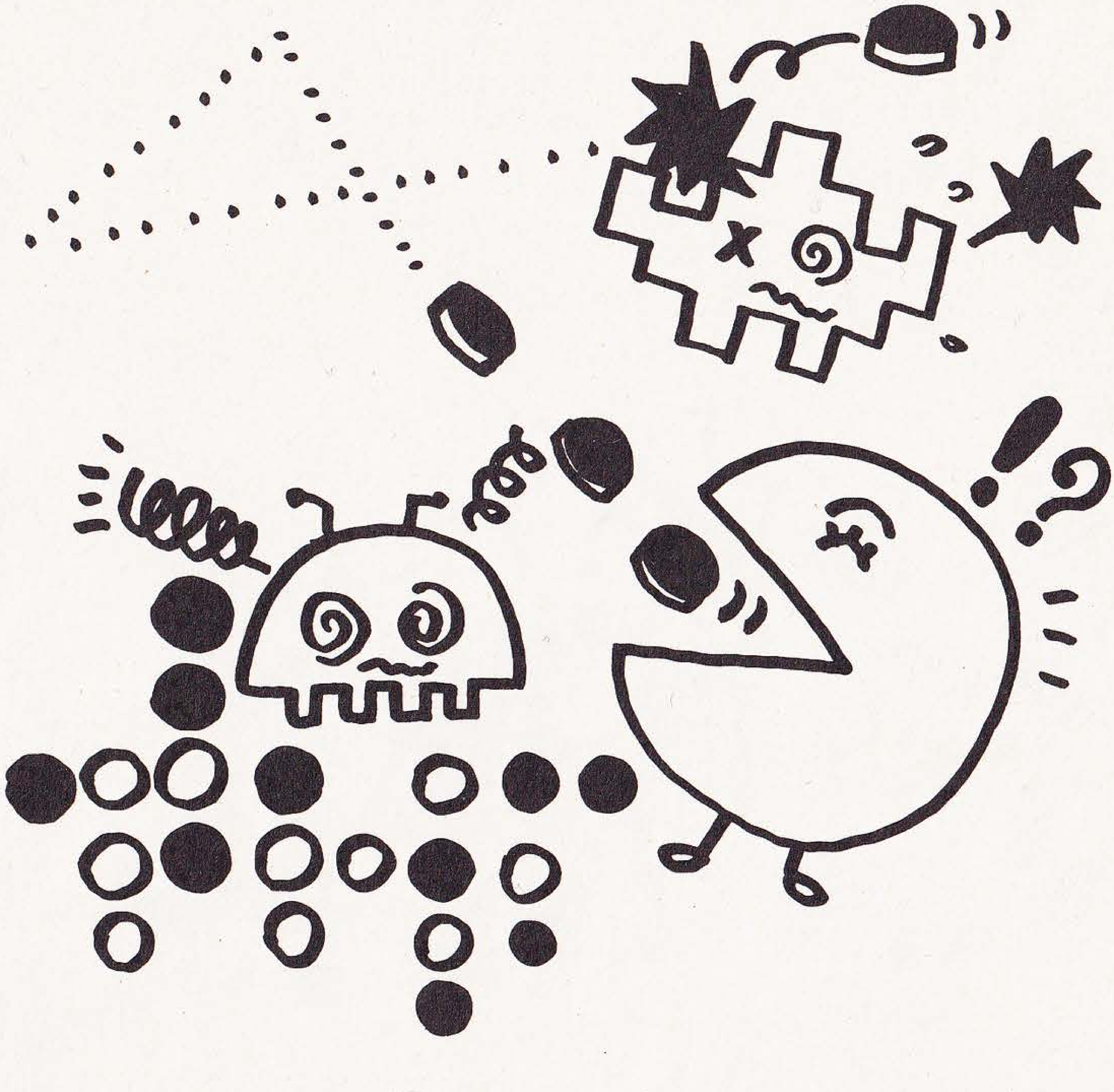
2．思考ゲーム(オセロ，麻雀など)

3．アドベンチャーゲーム

4．シミュレーションゲーム

5．ロールプレイングゲーム

6．運によるゲーム

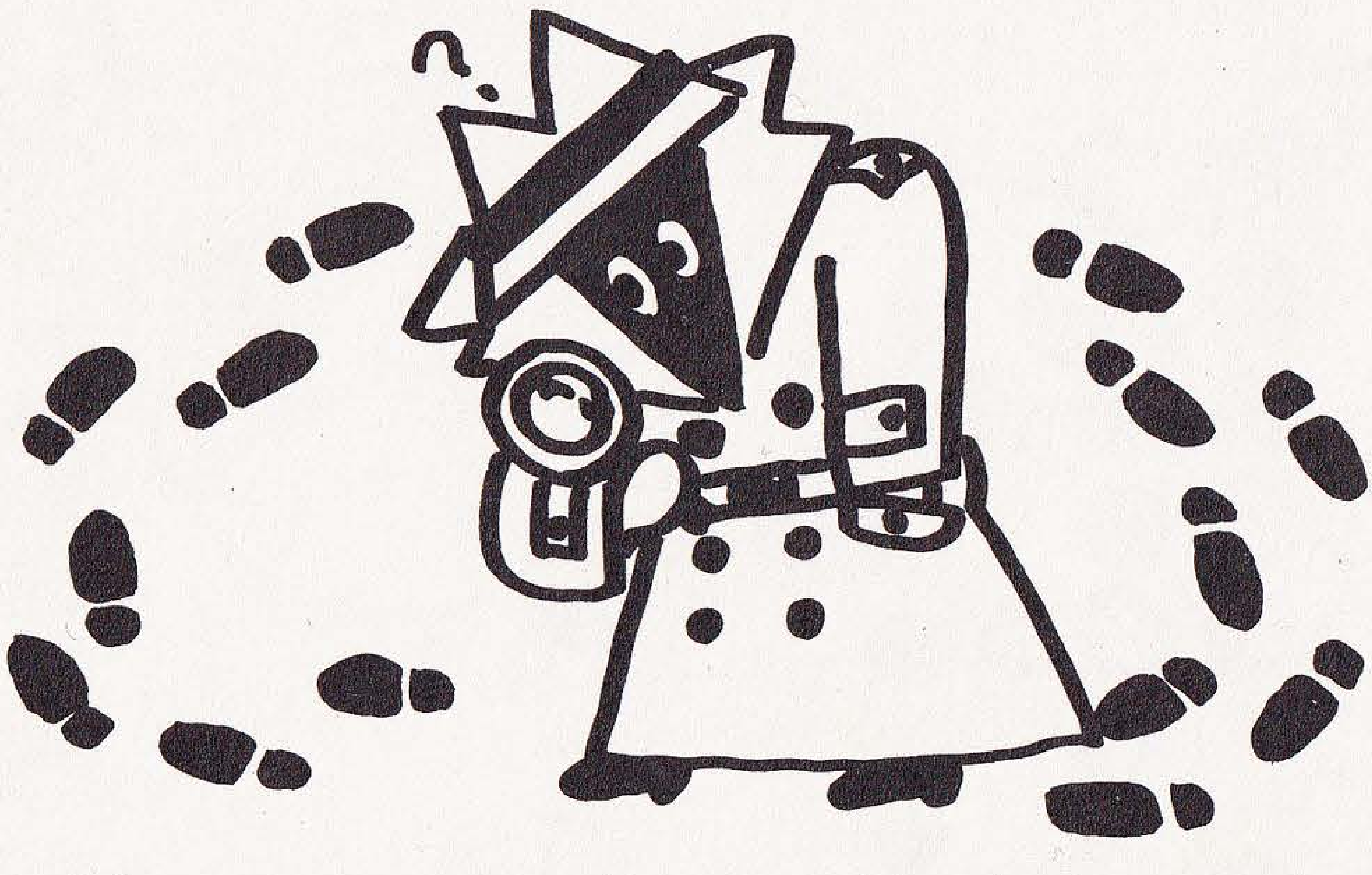


これらのうち，耳慣れないいくつかについて少し説明しましょう．

まずアドベンチャーゲーム．その名のとおり，冒険的要素(スリル，サスペンスetc.)を含んだ謎解きゲームです．ゲームの中で，「宝を探す」とか「地球を救う」といった使命を帯びているのが特長です．コンピュータに指令(単語とか簡単な文)を与えることでゲームは進行します．たとえば，北へ行けと命令するときは，キーボードからNORTH(北)と打ち込みます．そしてどんな命令を与えたらよいか，試行錯誤を繰り返しながら見つけていくのです．

次はシミュレーションゲームです．これは，メカ・マシーンの操縦や歴史的事件など，現実にある事柄をリアルに再現したゲームです．あたかも自分が飛行機のコックピットにいるかのように，スクリーンに景色が流れるフライトシミュレータとか，実際の戦争の展開を忠実に再現するウォーゲームなどがあります．

最後にロールプレイングゲーム．ロールプレイングとは「役割を演じる」という意味です．つまり「××ごっこ」というのを発展させたようなもので，ゲーム参加者はそれぞれの登場人物になりきって遊びます．現在のところ，ゲーム盤を使って遊ぶもの(ボードゲーム)のほうが主流で，マイコンゲームの世界にはまだ登場したばかりです．



さて，以上マイコンゲームを簡単に分類してみたわけですが，これから私たちが作ろうとしているのはアクションゲームです．アドベンチャーゲームやシミュレーションゲームなどはプログラムが非常に大きくなるので，この本では断念しました．

なお，マイコンゲームのジャンルはまだまだ増える余地があります．なにせマイコンの歴史はまだ浅く，その能力も日に日に進歩しているからです．もしかしたら本書をお読みのあなたが，新しいゲームの分野でパイオニアとなるかもしれません．

2節 ゲーム作りの一般的手法

短いプログラムなら，アイデアを思いついたその場でプログラム作りにとりかかっても，なんとかできてしまいます．ところがプログラムが大きくなると，その方法ではなかなか作れなくなってしまいます．あるプログラムを作るのに2時間かかったとして，それではその倍の大きさのプログラムが4時間でできるかというと，かならずしもそうはいきません．プログラムの質は同じとしても，現実にはさらに時間がかかるのが普通でしょう．

ゲーム作りにかかる時間というのは，プログラムの大きさに比例するものではありません．プログラムの大きさが単純に倍になったとすれば，それを矛盾なく作るためには，その何倍もの時間を要するのです．

さらに，いったん作ったプログラムのまちがいを修正する，いわゆるデバッグという作業も，プログラムが大きくなればなるほどやりにくくなっていくのです．

それではどうしたら大きなプログラムを短い時間で作れるでしょうか．その答えを一言で言うなら，構想から実際のプログラム作りまでを順序だてて行なう，ということにつきるでしょう．そこでこの手順を**ステップ1**から**ステップ6**までに分解して以下に述べました．

おもしろいゲームを作るためには，自分のアイデアをプログラムの上に上手に表現できるようにならなければなりません．このステップ1からステップ6までは大きなプログラムを手ぎわよく作るための方法を述べたものであるとともに，自分の考えをプログラムという形で表現するためのコツを述べたものです．



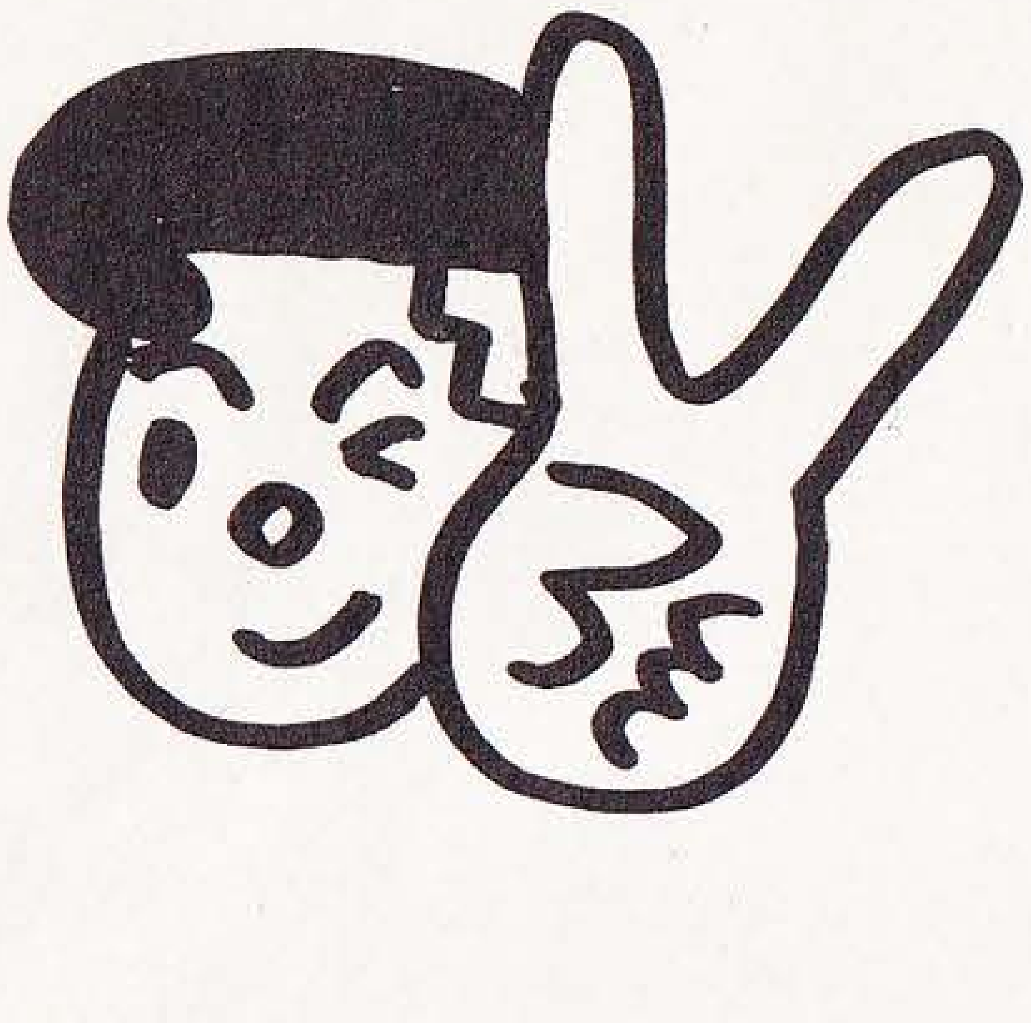
ステップ1 どんなゲームを作るのか決める

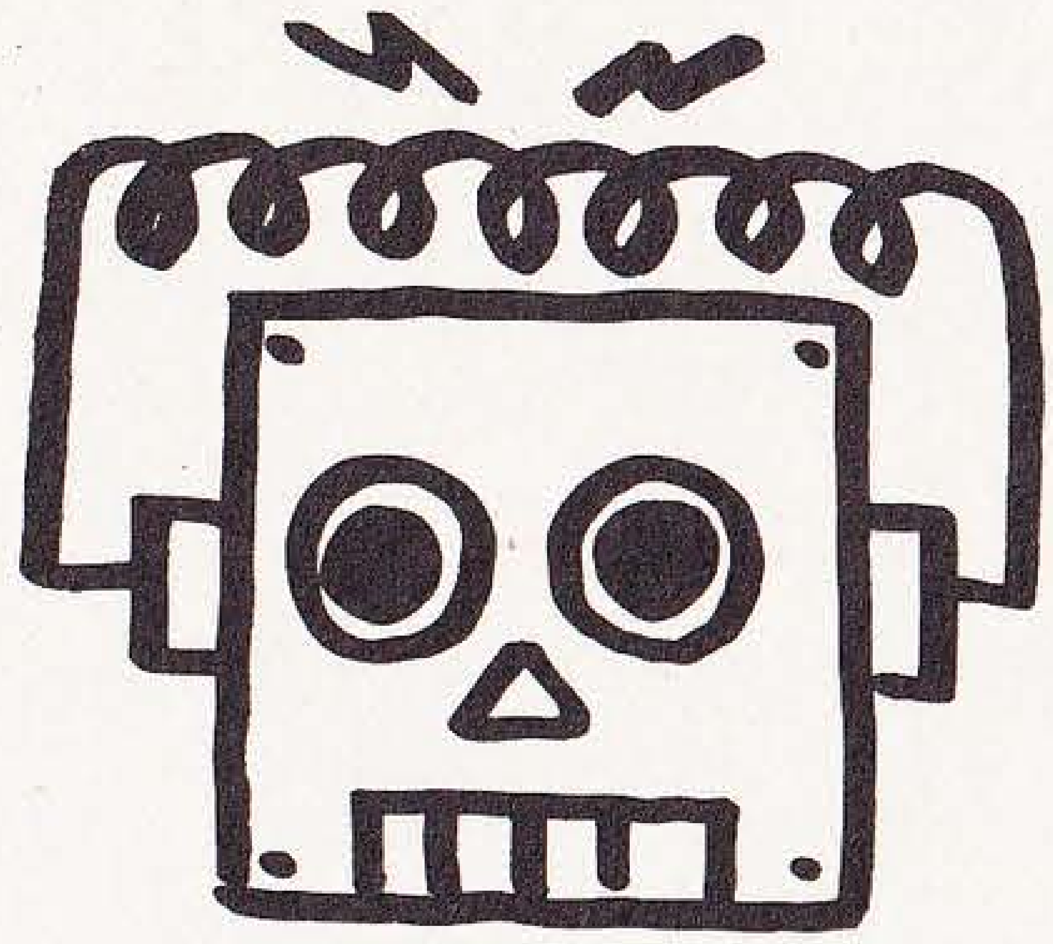
ゲームを作るには，まずアイデアが必要です．何かゲームを作りたいと思っても，どういうゲームを作るのかがはっきりしなければ仕方がないわけです．すでに知っているゲーム，たとえばブロックくずしを作ろうというのならば，アイデアは得られたことになりますが，自分で何か新しいものを作るためには，アイデアを出さなければなりません．

筆者は自分でゲームを作っているので，どうしたらよいアイデアが得られるのかとよく人から尋ねられます．実際にアイデアが浮かぶきっかけというのは，ゲームによりまちまちです．「ふっと何かが突然頭に浮かぶ」などということもありますが，たいていは何かを見て「これはゲームになる！」と感じる場合が多いようです．皆さんもその気になって探せば，「ゲームの種」は意外にどこにでもころがっているものです．

しかしそうは言っても，自分でゲームを作るのははじめてだという人には，アイデアをいきなり探すのは難しいかもしれません．そういう人たちには，すでにあるゲームを自分なりに作ってみることをおすすめします．すごろくゲームやオセロ(コンピュータに打つ手を考えさせるものはチト難しいが，人間対人間のものなら簡単)などを作ってみるのがよいでしょう．しかし，皆さんはせっかくこの本を手にしているのですから，この本に載っているゲームをまず試してみてから，それをもとにちょっと違ったゲームを作るところから始めるとよいでしょう．

そして，ある程度アイデアの求め方がわかり，プログラムの作り方もわかるようになったら，何かオリジナルなアイデアをもとに，新しいゲームをどんどん作ってみましょう．そして本当におもしろいゲームができ上がったら，ものはためしで雑誌に投稿したり，マイコンショップへ製品化の相談に行ったりするのもよいでしょう．





ステップ2 実現可能か検討する

マイコンの性能には限度があるので，プログラムを工夫すればどんなアイデアでも実現できる，とはかぎりません．たとえば，画面の上で何十個もの物体を自由自在に動かすプログラムを作ろうと思っても，マイコンの速度ではとても追いつきません．実際にそのプログラムを作ったとしても，非常にゆっくりとした動きにしかならないのです．

そこで，思いついたアイデアがマイコンで実現できるかどうか，よく考える必要があります．アイデアを思いついたら，まずそれを紙の上に書いてみましょう．そしてどんなプログラムを作ったらよいかざっと考えてみましょう．初心者のうちはどんなプログラムを作ったらよいかわからないので，ある程度の経験を積まなければなりません．この一冊を読んで実際にプログラムを作ってみれば，そのあたりはだいぶわかるようになるでしょう．

さて，検討の結果プログラム実現のめどがつけば，メデタシメデタシというわけですが，そうでなかった場合はどうすればよいでしょうか．本当にどうしても実現不可能なものならば，あきらめるしかありません．しかしここでのネバリが実は大切なのです．つまり，なんとか工夫すればアイデアを実現できないか，とあれこれ試みることが大事なのです．たとえばさきほど，何十もの物体を画面の上で自由自在に動かすのは無理だと言いました．しかし，これにしてもまず数個の物体を動かし，次は別の数個を動かし……というようにすれば，一度に全部を動かした場合と同じように見せることができるでしょう．またインベーダーゲームのように，動く方向がどのインベーダーも同じならば，全部の移動方向をべつ

べつに計算する場合に比べて，かなり速い動きを実現できるでしょう．このように，いわば「ゴマかす」ことにより，あるいはアイデアに修正を少し加えることによって，もとのアイデアに近いものを実現することが多くの場合可能なのです．

さて，このようにアイデアに多少の修正を加えて，おおよその実現の見込みをつけたものを「構想」と呼ぶことにしましょう．すると結局ステップ1，2は，

ステップ1：アイデアを得る

ステップ2：構想を作る

ということになるわけです．

ステップ3 構想をマイコン上の表現に置き換える

構想というのは，プログラムの実現可能なことをおおよそ確かめたものだと述べましたが，ここではもう一歩進んで，構想をマイコンに理解できる表現に置き換えようというわけです．「マイコンに理解できる表現」といってもわかりにくいかもしれません．具体的にはどういうことをさすのか，述べていきましょう．

マイコンゲームでは，画面の上を何かが動いたり，文字が出たり，絵を描いたりといったことはまず不可欠の要素でしょう．そこでプログラムの中で画面に文字・絵を表示する必要が出てきます．もしもこの表示の方法がひと通りしかなければ何も考えなくていいのですが，実際にはいくつもの方法が考えられるので，どの方法をとるのかをはっきりさせる必要が出てきます．

まず作りたいゲームの画面の模様を紙に書いてみましょう．スクリーン・レイアウトシートという専用の紙も市販されていますが，方眼紙があれば十分です．マイコンの画面の点は(この点のことをドットとかピクセルとか呼びます)，MSXでは画面が横256×縦192に分割されていますから，1mm方眼紙なら横25.6cm，縦19.2cmの長方形の枠を作れば，1つのマスが画面上の1つの点に対応するようになります．そして枠の中に表示したい絵や文字を書いてみるのです．これを「画面のレイアウトをする」と言います．レイアウトを行なわないと，画面のバランスが悪くなりがちです．

次にこのレイアウトをもとに，それをマイコンの言葉(この本で扱う範囲ではBASIC)ではどう表わしたらよいかを考えていきます．その際にまず基本となるのはスクリーンモードです．MSXでは4つのスクリーンモードがあり，それぞれ次のような特徴を持っています．

スクリーンモード

モード0……画面に横40×縦24文字まで出せる．文字しか出せず，また色は1色しかつけられない．グラフィックス命令・スプライトは使えない．

モード1……画面に横32×縦24文字まで出せる．VDP(注)を操作すれば複数の色も出せるが，普通は1色しかつけられない．スプライトは使えるが，グラフィックス命令は使えない．

モード2……横256×縦192ドットでグラフィックス命令が使える．色は，横方向に並んだ8ドット単位でつけられる．スプライトも使える．

モード3……横64×縦48ドットでグラフィックス命令が使える．色は各ドットごとにつけられる．スプライトも使える．

(注)VDP：ビデオ・ディスプレイ・プロセッサの略で，テレビ画面の状態の操作を行なうLSI.

ここで，グラフィックス命令というのは，線を引いたり，点を打ったり，円を描いたりする命令群(LINE,PSET,CIRCLEなど)のことをさしています．またVDPを操作するというのは，本書で扱う範囲外なので詳しくは述べませんが，BASICのBASE,VDP,VPEEK,VPOKEの4つの命令を使うと，いろいろと変わったことができるのだと考えてください．

さて話を戻して，いま述べたような各スクリーンモードの特徴を考えて，自分の作ろうとしているゲームにはどのモードを使うか決めるわけです．たとえば文字ばかりが出てくるゲームなら**モード0**か**1**がよいでしょう．細かい線や絵を使うのなら**モード2**．ドットは粗くてもいいが，各点それぞれに色をつけたければ**モード3**という具合です．

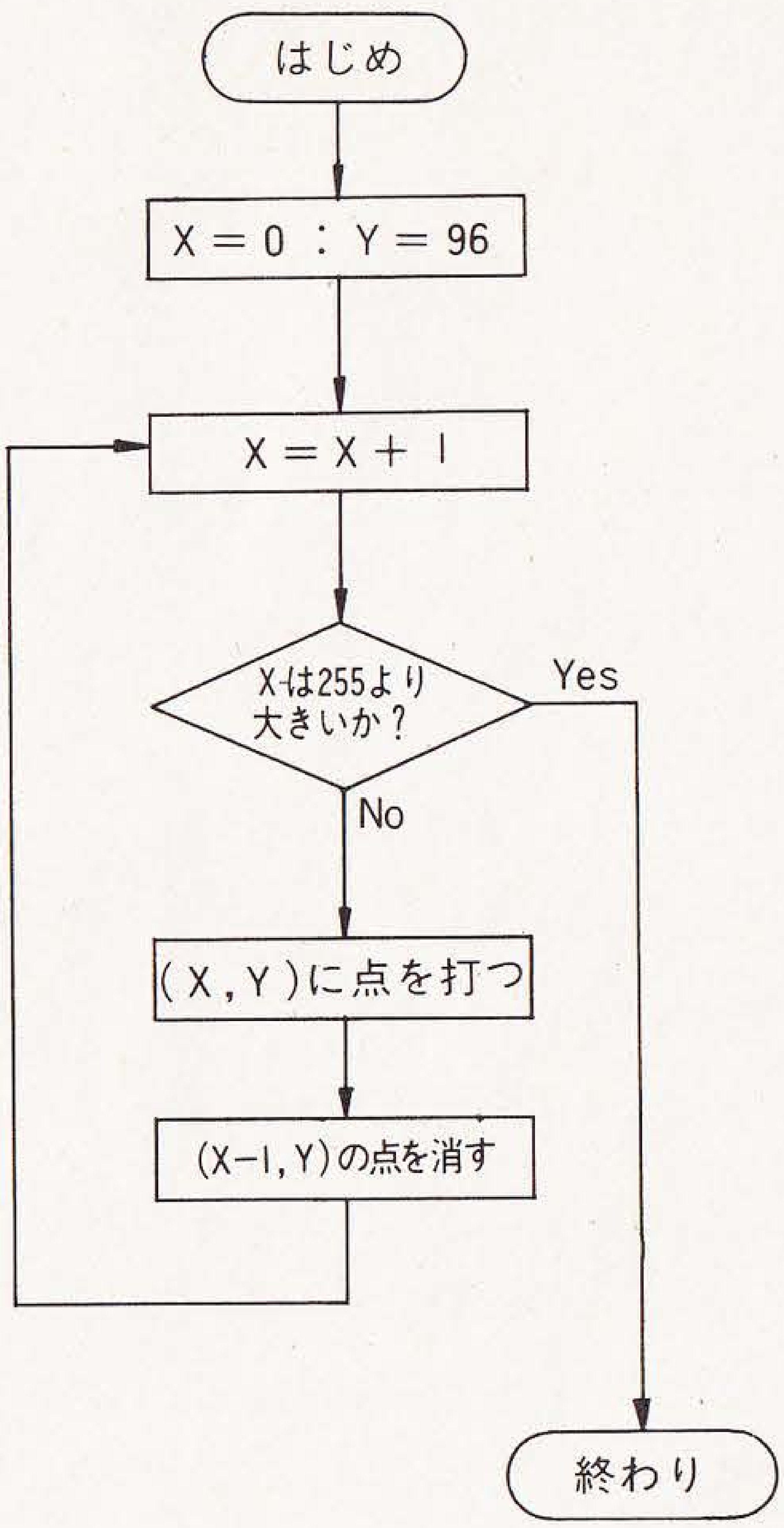
スクリーンモードを決めたら，次にレイアウト用紙に描いた絵をどう表現すればよいかを考えます．ここはLINE命令で線を引けばよいだろう，とかこのインベーダーはスプライトを使って動かそう，というようにです．

以上が，画面の構成に関しての「構想をマイコン上の表現に置き換える」，ということの具体的内容です．初心者の人やMSXの機械を持っていない人は，スプライトとかグラフィックス命令という言葉を聞いてまごついたかもしれません．し

かし，ここでは流れだけを理解できれば十分です．ここで，よくわからなかった人も２章以降の実践編のほうを読めば疑問は解決することでしょう．

さて次に画面以外の部分について述べてみましょう．構想の段階で，実現可能な見通しを持っておくということは述べました．次の段階ではそれをさらに進めて，具体的にどういう論理を使えばよいのか考えてみよう，というわけです．論理などというと「ワッ，難しそう」と思う人もいるかもしれませんが，恐れることはありません．数学や論理学の難問を解こうというのではないのです．例として，画面の上で１つの点を左から右へ動かすというときに必要な論理を考えてみましょう．**図 1** を見てください．これはフローチャートというもので，作業の手順を示したものです．論理を表わすのに，ここではフローチャートを使って説明します．

図 1



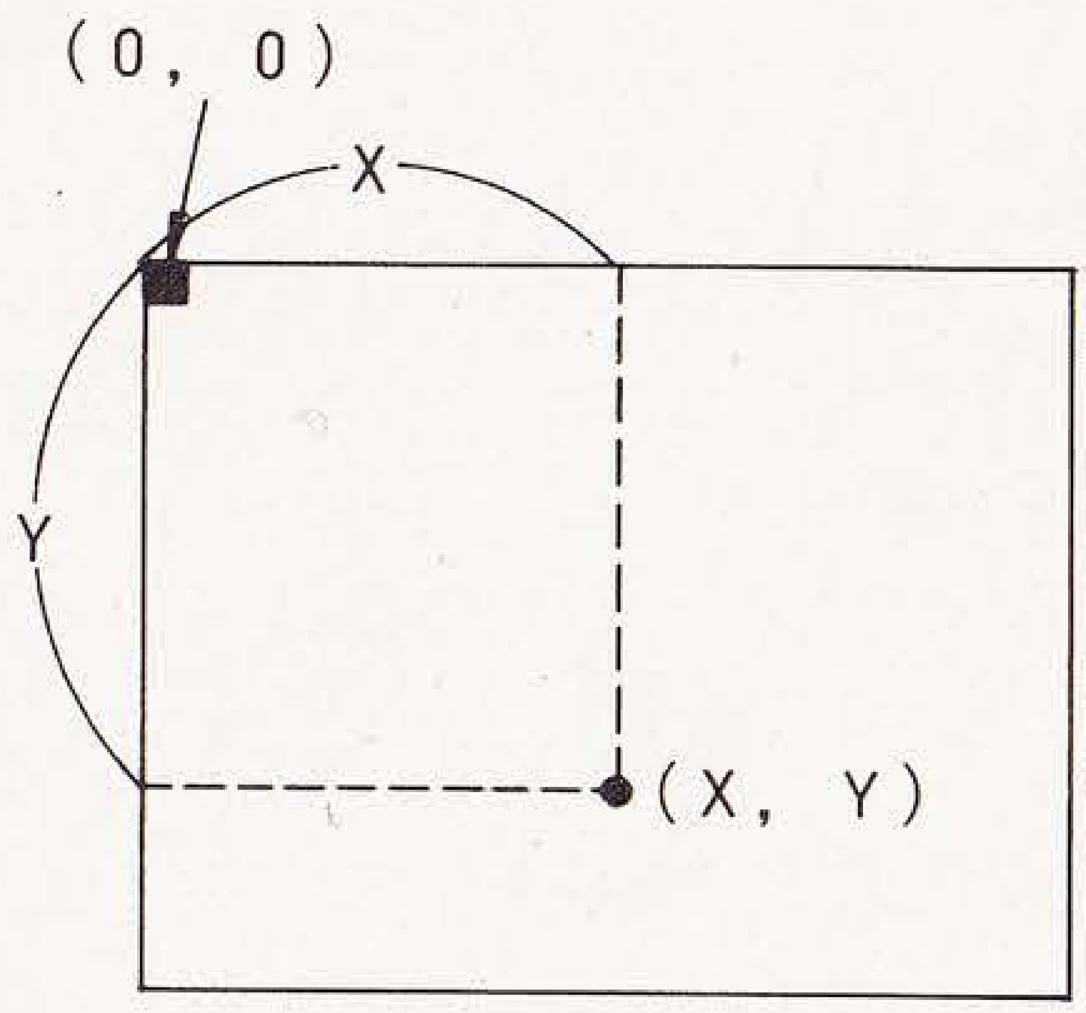


まず最初にX＝０：Y＝96と書いてありますが，これはXを０にし，Yを96にするということです．次にX＝X＋１とあるのは，「XをX＋１にする」つまりXを１

つ増やすという意味です．次にひし形の中に1つの疑問文が書いてありますが，これが正しければYesのほうに，まちがっているのならNoのほうに進むということをさしています．この場合，Yesならば終わりになり，Noならばさらに下に続くことになります．

次に,(X,Y)に点を打つとありますが，これは画面の左上すみを(0，0)と考えて，右にX番目，下にY番目の場所に点を打つということです(図2)．

図2





これだけのことを知った上でフローチャートをじっとにらむと，このフローチャートが，1つの点を画面上で左から右へと動かすものであるというのが見えてくるでしょう．

論理を組み立てるというのは，「点を動かす」というこの例のように，一連の動作を簡単な部分に分割するということです．この例のような場合，そんな作業をしなくてもプログラムを作れるかもしれませんが，複雑なものになると，思わぬ勘ちがいをしたままプログラムを作りはじめてしまう危険があります．まずはじめに論理を組み立てることが大事になってくるわけです．

ステップ4 プログラムの姿をはっきりさせる

中規模以下のプログラムならば，ステップ3までを行なうだけでだいたいプログラムの姿が見えてきますが，大きなプログラムの場合には，まず全体の大きな流れ図(ゼネラルフローチャートと呼ぶ)を作っておくと便利です．この節の最初のほうで，プログラムが大きくなると，大きくなった割合以上に作る労力は大きくなると述べましたが，きちんとしたゼネラルフローチャートを作って全体の流れを分割しておけば，小さなプログラムを多数作るのと同じことになり，労力は

そんなに増えずにすむわけです．このようにプログラムを小さなまとまりごとに分割して作る方法を，モジュール化プログラミングと呼びます．そして，分割された1つのまとまりのことをモジュールと呼びます．

まだこれだけの説明ではわからない人もいると思いますが，2章以降の実例を見ればだんだんにわかると思います．

なお，ステップという名前をつけて，作業の過程を順に述べていますが，実際には常にこの順に作っていくのが最善というものでもありません．とくに本当に大きいプログラムにおいては，ゼネラルフローチャートを固めてから，ステップ3の作業にはいるということも多いでしょう．

ステップ5 プログラムを組む

さて，いよいよ実際にプログラムを作っていく段階にはいるわけですが，プログラムの組み方に関しては，一般的にアドバイスできることはあまりありません．ステップ4で述べたように，「モジュール化することを心がけよ」などといったことは言えますが，1つのモジュール内でのプログラムの組み方となると，BASICのコマンド(命令)の1つ1つの使い方が問題になってきます．またプログラミングのスタイルというのも各人いろいろで，紙の上にプログラムを書いてよく考えてからマイコンに打ち込む人もいれば，そんなことはせずにいきなりキーボードに向かう人もいるという具合で，どれがよいとは一概に言えません．

そんなわけで，大事なのはある程度の知識と慣れです．どの程度の知識があればいいのかということは，次の3節で述べることにします．また慣れということについて言えるのは，とにかくどんどんプログラムを作ってみること．初心者のうちからいきなりまちがいのないプログラムを作れる人間なんているはずがないのです．それどころか，どんなに慣れたプログラマーだって，大きなプログラムを一度で動かす芸当はまずできないのです．

だからまったくウマくいかなかったからといって，がっかりしてはいけません．その原因を見つけることが大切なのです．ウマくいかなかった原因を見つければ，それだけ着実に一歩前進するのですから——．

ステップ6 プログラムをなおす

ステップ5で述べたように，はじめて組んだプログラムが思いどおりに一度で

動くというのは，ほとんど奇跡に近いことです．そこでデバッグ，つまりプログラムをなおす作業を効率よく行なうことが大切になってくるわけです．

BASICのプログラムを実行したとき，バグ(プログラムのまちがい)は次の2つのうちのどちらかの形で検出されます．

1．エラーが出てプログラムの実行が停止される．
2．エラーは出ないが，期待どおりの動作をしない．

このうち1の場合ならば，エラーの出た行を手がかりにして調べていけば，だいたい解決がつきます．それに対して2の場合には，どこがおかしいのかわかりづらく多少やっかいです．解決法としては，予期しなかった動作の起こり方とタイミングをもとに，プログラムリストで調べていく以外ありません．プログラムがきちんとモジュール化されていれば，どこでおかしな動作を起こしたかを予想するのが容易になります．またきちんとしたフローチャートを作っておけば，それも解決の糸口となるでしょう．

大きなプログラムともなると，画面にリストを表示させながらまちがった場所を見つけていくのは大変です．もしもプリンターを持っているならば，リストを紙の上に打ち出して，紙の上でバグの場所を探すことをおすすめします．

まとめ

以上，ゲームプログラムを作るのに必要な手順をステップ1から6までにわけて述べましたが，これはプログラミングの基本テクニックであって，人により場合によりさまざまに違ってくるものです．

本当におもしろいゲームを作るためには，場当たり的なやり方によるのではなく，やはりステップを踏むことを覚えるのが大事だと心得ておいてください．

3節 ゲーム作りに必要な知識

前節ではゲームを作るためには，どういう手順をとるのがよいのかを述べました．しかし，いくらそれがわかったところで，BASIC言語についてある程度の知識がないことには，自分でゲームを作ることはできません．

あのぶ厚いマニュアル(文法説明書)を十分に理解するためには，かなりの努力

重要度別BASICコマンド一覧

コマンド・ステートメント	ランク	主な用途
CLOAD	A	
CLS	A	7
COLOR	A	7
CSAVE	A	
END	A	
FOR～NEXT	A	
GOSUB	A	
GOTO	A	
IF～THEN～ELSE	A	
INKEY$	A	
INPUT	A	2
LET(＝)	A	
LIST	A	
LOAD	A	
PLAY(ステートメント)	A	
PRINT	A	2
REM	A	
RETURN	A	
RUN	A	
SAVE	A	
SCREEN	A	
AND	B	4
ASC	B	
CHR$	B	
CONT	B	
DATA	B	
DIM	B	
DRAW	B	7
INT	B	
KEY ON/OFF	B	
LINE	B	2
LOCATE	B	2
MOD	B	
OR	B	
PAINT	B	7
PSET	B	7
PUT SPRITE	B	7
READ	B	
RENUM	B	
RND	B	

コマンド・ステートメント	ランク	主な用途
SOUND	B	
SPRITE ON/OFF/STOP	B	7
SPRITE$	B	7
WIDTH	B	2
ABS	C	
BEEP	C	
CIRCLE	C	7
CLEAR	C	
COS	C	6
DEF INT/SNG/DBL/STR	C	
INTERVAL ON/OFF/STOP	C	
KEY(n) ON/OFF/STOP	C	
LLIST	C	1
LPRINT	C	1
NEW	C	
ON ERROR GOTO	C	
ON GOTO	C	
ON GOSUB	C	
ON INTERVAL GOSUB	C	
ON KEY GOSUB	C	
ON SPRITE GOSUB	C	
ON STOP GOSUB	C	
ON STRIG GOSUB	C	8
PLAY(関数)	C	
POINT	C	7
PRINT USING	C	2
RESTORE	C	
SIN	C	6
SQR	C	6
STICK	C	8
STOP ON/OFF/STOP	C	8
STRIG	C	8
STRIG ON/OFF/STOP	C	6
TAN	C	6
TIME	C	
TRON/TROFF	C	
AUTO	D	
BASE	D	4
BLOAD	D	5

コマンド・ステートメント	ランク	主な用途
BSAVE	D	5
CDBL	D	
CINT	D	
CLOSE	D	
CSNG	D	
CSRLIN	D	
DEF FN	D	
DELETE	D	
ERL	D	
ERR	D	
EXP	D	6
FIX	D	
FRE	D	
HEX$	D	5
INPUT#	D	3
INSTR	D	2
KEY	D	
KEY LIST	D	
LEFT$	D	2
LEN	D	2
LOG	D	2
MAXFILES	D	3
MERGE	D	
MID$(関数)	D	2
MID$(ステートメント)	D	2
NOT	D	
OPEN	D	3
PAD	D	
PDL	D	8
PEEK	D	5
POKE	D	5
POS	D	2
PRESET	D	7
PRINT#	D	3
RESUME	D	
RIGHT$	D	

コマンド・ステートメント	ランク	主な用途
SGN	D	
SPACE$	D	2
STOP	D	
STRING$	D	2
STR$	D	
TAB	D	2
VAL	D	
VDP	D	4
VPEEK	D	4
VPOKE	D	4
XOR	D	
ATN	E	6
BIN$	E	
CALL	E	
DEF USR	E	
EOF	E	3
EQV	E	
ERASE	E	
ERROR	E	
IMP	E	
INP	E	
INPUT$	E	3
LINE INPUT	E	
LINE INPUT#	E	3
LPOS	E	
LPRINT USING	E	3
MOTOR	E	7
OCT$	E	7
OUT	E	
PRINT# USING	E	3
SPC	E	2
SWAP	E	
USR	E	
VARPTR	E	
WAIT	E	

ランクA：絶対知っているべき
ランクB：知っているべき
ランクC：知っていたほうがよい
ランクD：できれば知っておく
ランクE：知らなくてもよい

用途
1…プリンタを持っている人が使うもの．2…いろいろなメッセージを画面に出すようなゲームに関係するもの．3…ファイルを扱うもの．グラフィック画面に文字を書いたり，カセットを扱うのに使う．4…VDPを扱うもの．画面に特殊な形で文字や絵を出すのに有効．5…機械語に関連するもの．6…計算をするのに使うもの．7…グラフィックス命令．8…ジョイスティックやパドルを持っている人に有効なもの．

を要します．ゲームを作るためには，BASICの命令をなるべく多く知っていたほうがよいのは確かです．しかし，全部の命令を頭に入れておかなければゲームは作れないというわけでもありません．そこでBASICの命令を，筆者の独断と偏見により重要度別(ゲームを作るに際しての)にした表を作ってみました．命令を「覚える」というと，その書式から全機能に至るまで完全に暗記しなければいけないように思いがちですが，その必要はありません．おおよそどんな働きをするかを頭に入れておけば，マニュアルを引き引きプログラムを作っていくことができます．

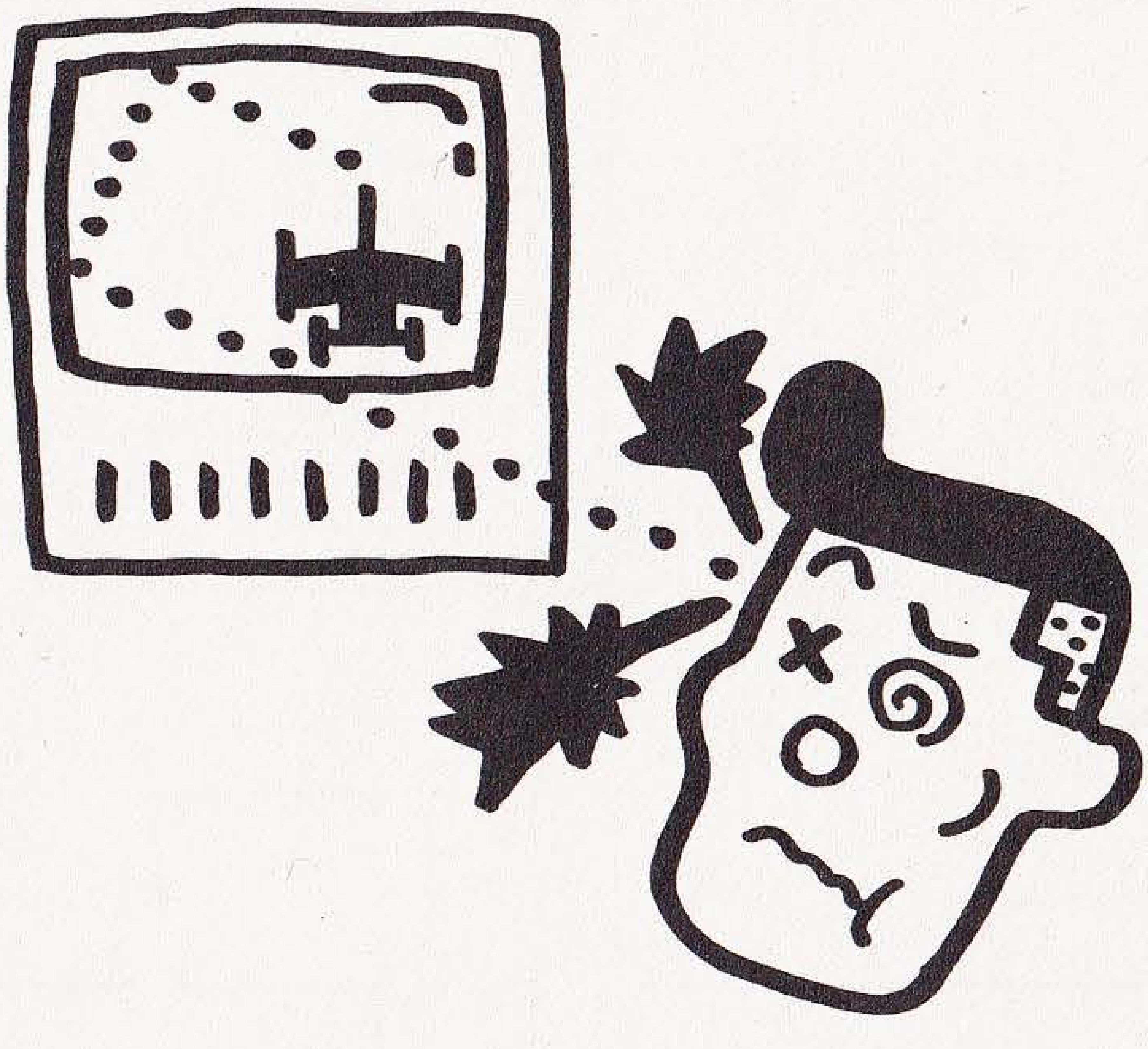
さて，BASICの命令についての心得はこれぐらいにして，ゲーム自体について多少お話ししましょう．ゲームプログラムを作る人には，それぞれの「流儀」みたいなものがあって，プログラミング・スタイルはまちまちです．しかし他人のよいところを取り入れる，というのはよいことです．ましてゲーム作りの初心者にとっては，他の人の作ったプログラムを参考にすることは大いにすすめられます．そのためには，なにも市販のソフトを何本も買わなくても，マイコン雑誌に載っているプログラムリストを利用すればよいのです．雑誌により難易度には多少の違いがあるので，本屋で自分の気に入ったものを買って，研究してください．

では，次章から実際にいくつかのゲームを作ってみることにしましょう．



2

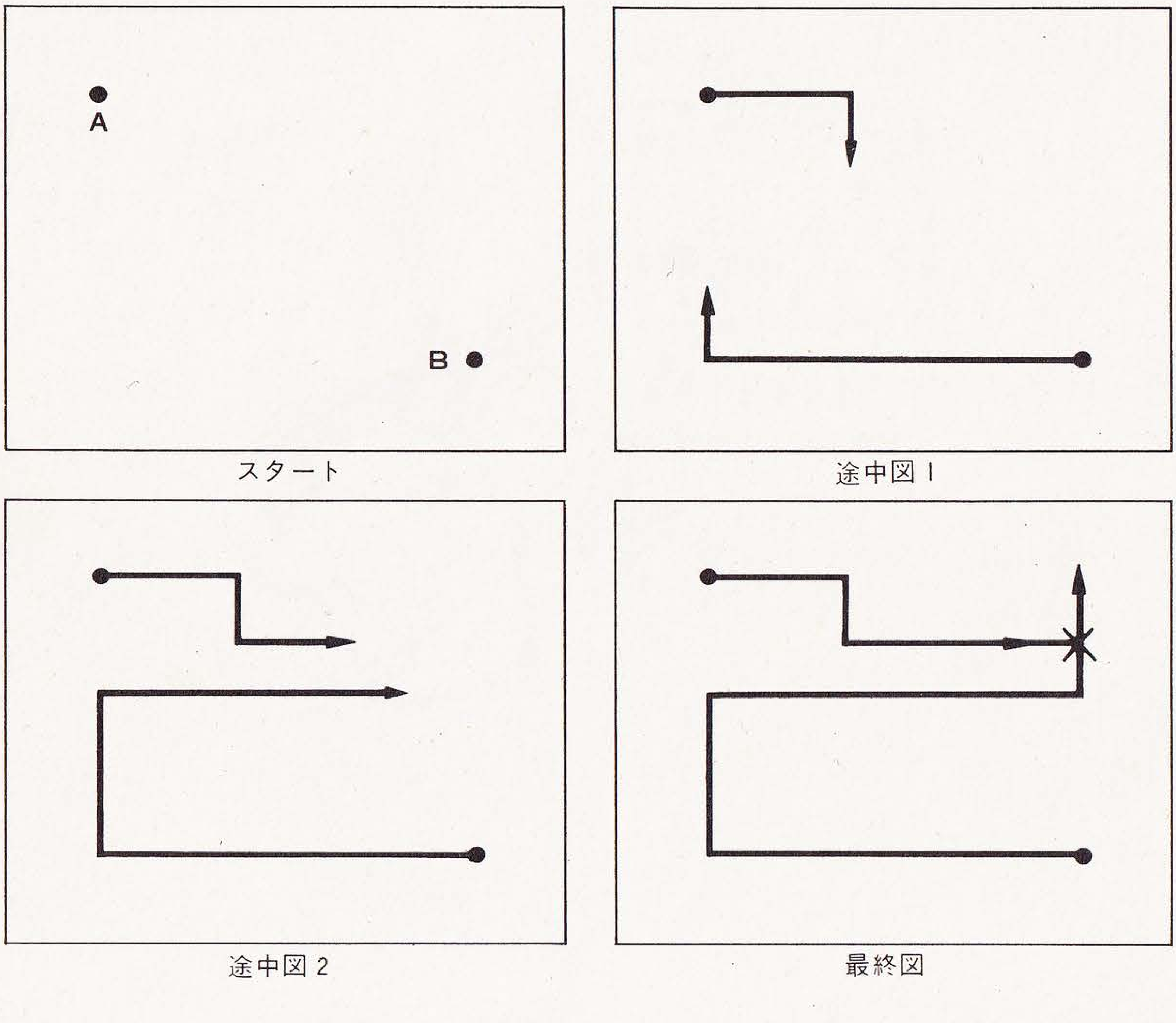
（実践編 I ）
バリケードゲーム



1節 バリケードゲームとは

アーケードゲームのはしり，ブロックくずしが大流行していたころに，同じくらい人気のあったゲームがこのバリケードゲームです．2人で遊ぶゲームですが，ルールは単純です．2人がそれぞれ1つの点を受け持ち，それを上下左右に動かして壁を作っていきます．そしてどちらかが壁にぶつかったら，ぶつかったほうの負けとなります．点を動かす方向はキーによって操作します．**図3**では，A点からスタートした人が壁にぶつかったので，Bの人の勝ちになっています．

図3



このようにルールは簡単なのですが，キーを押して曲がるタイミングが勝敗を

微妙にわけるので，２人で遊ぶとけっこうアツくなります．また，相手の行く手をうまくはばんで自分の点を進めるのが，このゲームに勝つためのポイントです．慣れてくると，いろいろなカケヒキが出てきます．

2節 ゲームの骨組みを作る

アプローチ1 レイアウトを決める

前章２節のステップ２までの部分，つまり構想までの段階はこのゲームの場合は省略して，ステップ３の「構想をマイコン上の表現に置き換える」というところから述べていきます．

ゲームの内容は前節で紹介したとおりです．これをもとに，まずは画面のレイアウトから考えていきましょう．

「点が動いて壁を作っていく」わけですが，これをどう表現するか，その方法は大きくわけると次の２つです．

①“点”をグラフィックスで表わす
②“点”を文字(キャラクタ)で表わす

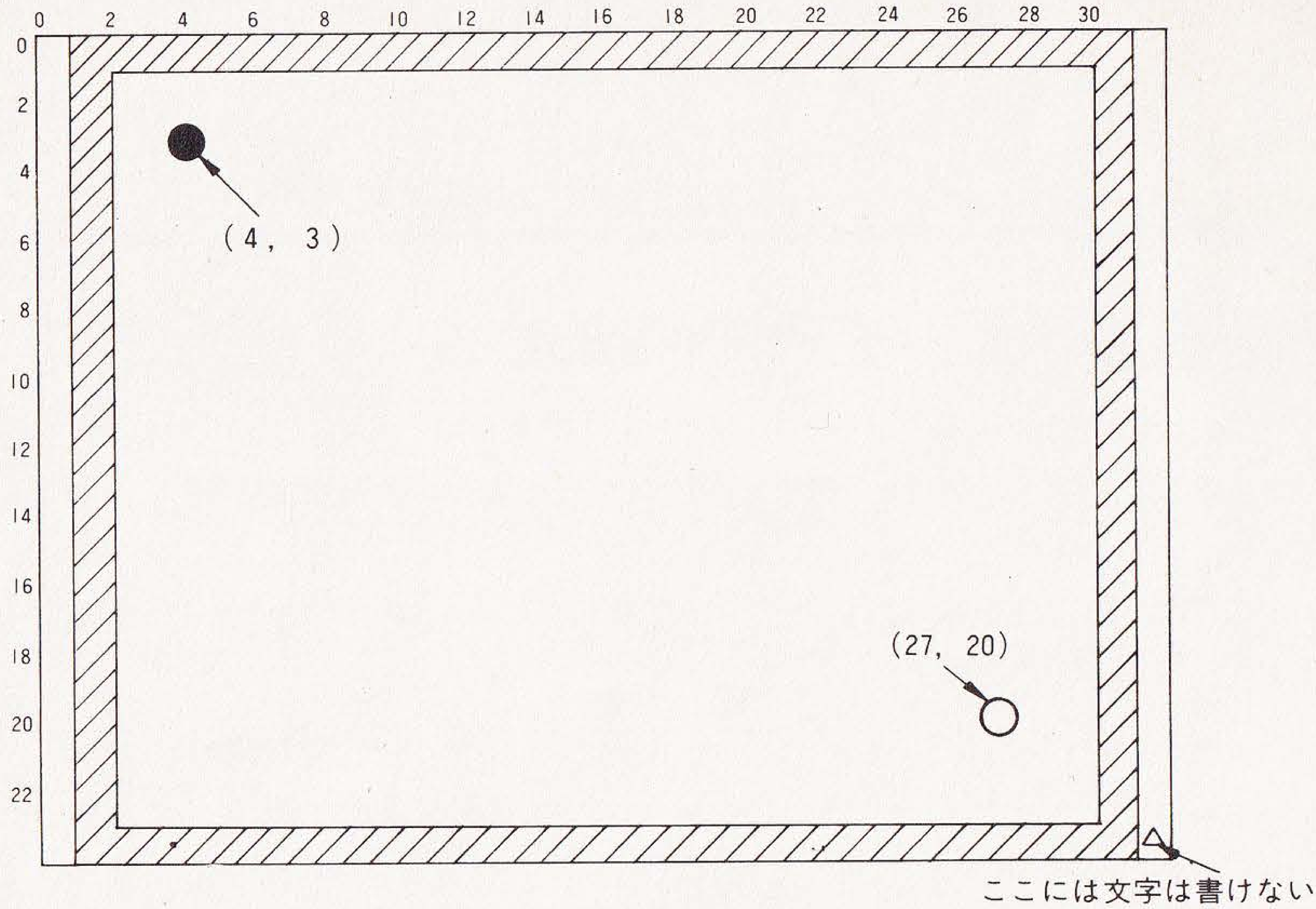
今回は後者の「文字」を使うことにしました．横32×縦24文字の**スクリーンモード１**を使うことにします．この理由は，グラフィックスを使うのに比べて文字のほうがプログラム作りが多少ラクになるからです．また横32文字のモードにすれば，縦，横の長さの割合が横40文字のモードよりも正方形に近くなります．

さて，それでは画面の様子を考えてみます．**図４**を見てください．細い枠になっているのが画面全体で，枠の外に書いてある数字は座標です．

まず外側の壁の位置は，左右とも１文字分だけあけ，上下はメイッパイに使うことにしました．１文字分あけた理由は，左右の画面が見づらいということもありますが，右下隅に文字を書くと，画面全体が１行上にズレてしまう(スクロールするという)からです．

次に点のスタート位置ですが，壁の左上隅と右下隅から２文字分離れたところにとることにしました(これはとくに理由はない)．そして２人のプレイヤーの点を，黒丸と白丸で区別することにしました．

図4



アプローチ2 主要部分の論理を考える

1．点の動かし方

さて2人のプレイヤーの点(黒丸と白丸)をどう動かしたらよいか考えましょう．

バリケードゲームの点の動きは，キーが押されないかぎり同じ方向へ進み，曲がる方向のキーを押すと進路を変えます．それに対応するキーは，機械に向かって右側のプレイヤーは，上下左右を↑，↓，←，→で表わし，左側のプレイヤーは，W，Z，A，Sで表わすことにします．

それでは，点が上や左へ動くのをどう表現するか？――プレイヤーの点がある場所の座標を(X, Y)，点の動く方向を(U, V)としてみましょう．X，Yはそれぞれ横座標，縦座標を表わすものとし，U，Vはそれぞれ1回に動く左右方向，上下方向の動きを表わすことにします．たとえば，(20, 10)というところで，左へ進む点があったとしたら，X＝20，Y＝10，U＝－1，V＝0となるわけです．

すると，毎回のこの点の位置(X, Y)は，

X＝X＋U：Y＝Y＋V

という計算を繰り返せば求められることになります．そして，UとVをキーが押されるごとに変えれば，バリケードゲームの点の動きが作れます．たとえば[↑]のキーが押されたら，

U＝０：V＝－１

とするわけです．

２．衝突の判定法

次に点と壁が衝突したかどうかを調べる方法を考えてみましょう．人間が調べるのであれば，目で見て「ぶつかった」，「いやぶつかっていない」などと言えばよいのですが，プログラムの場合はそう単純にはいきません．

方法としては，人間の目と同様に，実際の画面上に点があるかどうかを調べることを考えます．スクリーンモード２あるいは３であればPOINT関数を使って色を調べる方法もあるのですが，モード１では残念ながらPOINT関数は使えません．V―RAM(画面の情報が保存されている記憶素子)を直接VPEEK関数を使って調べてもよいのですが，V－RAM関係の説明は複雑になるので，ここでは使わないことにします．

そこで配列変数を使って調べる方法を採用することにします．B(X，Y)という配列を使い，その場所が壁ならば，

B(X，Y)＝1

とし，壁でなければ，

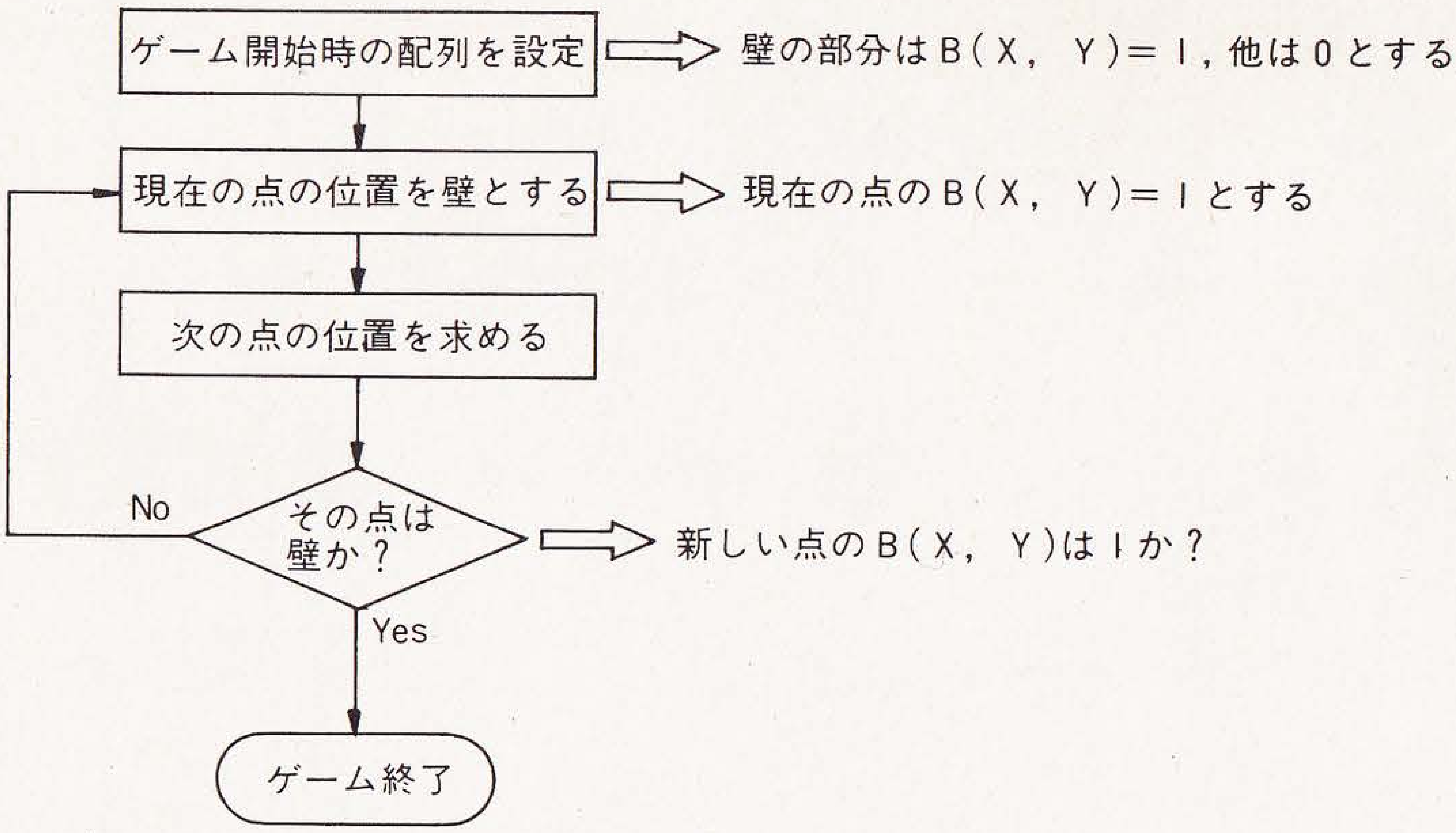
B(X，Y)＝0

というように取り決めます．たとえば(20，10)という場所が壁であるかどうかを調べるには，B(20，10)の値を求めればよいわけです．

ただ，この方法では画面全体の情報を得るのに700個以上の変数を用いることになり，メモリーの不足を心配する人もいるかもしれませんが，全体のプログラムが短いので大丈夫です．

衝突の判定をフローチャートで書くと，**図5**のようになります．

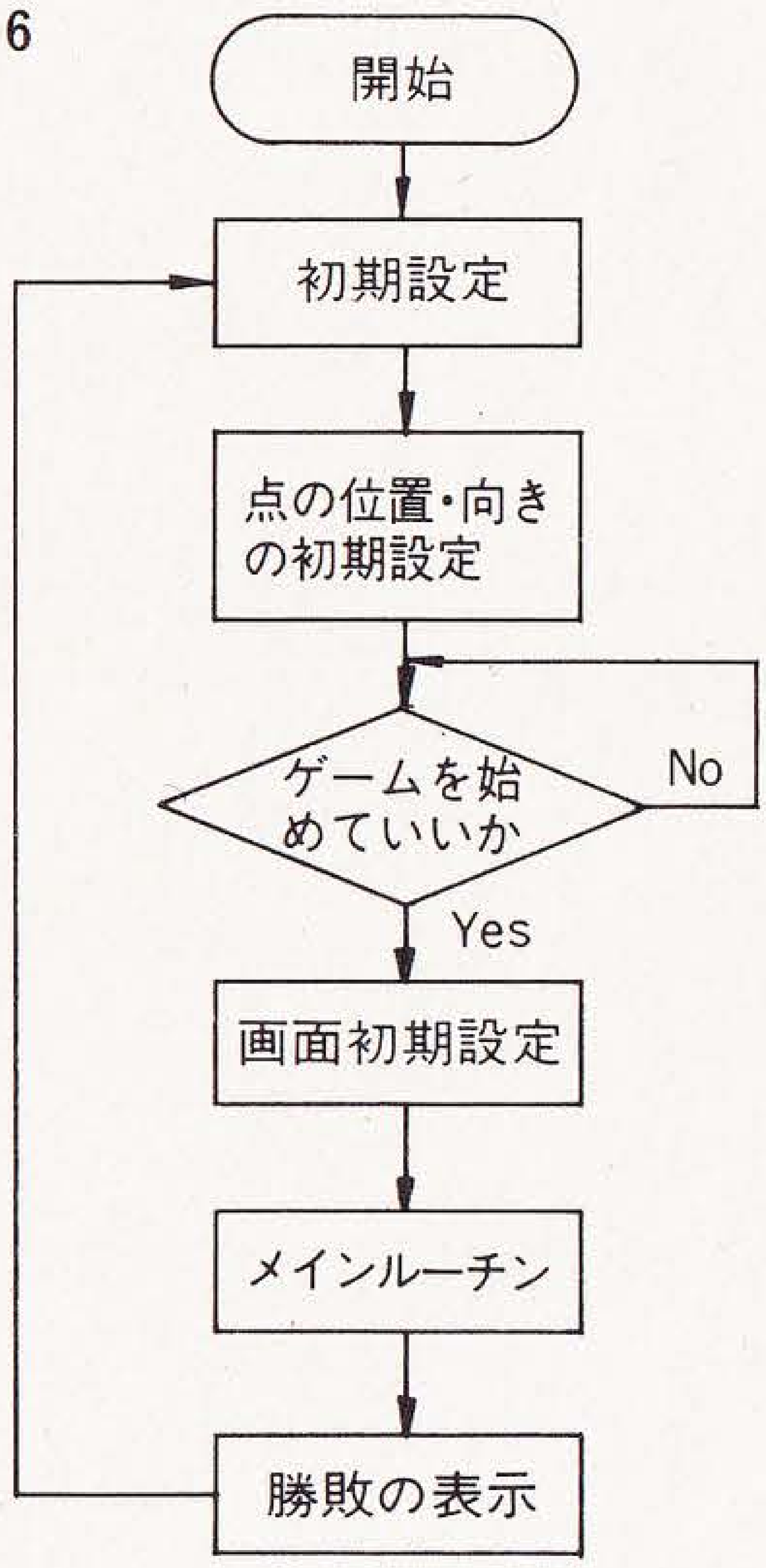
図 5

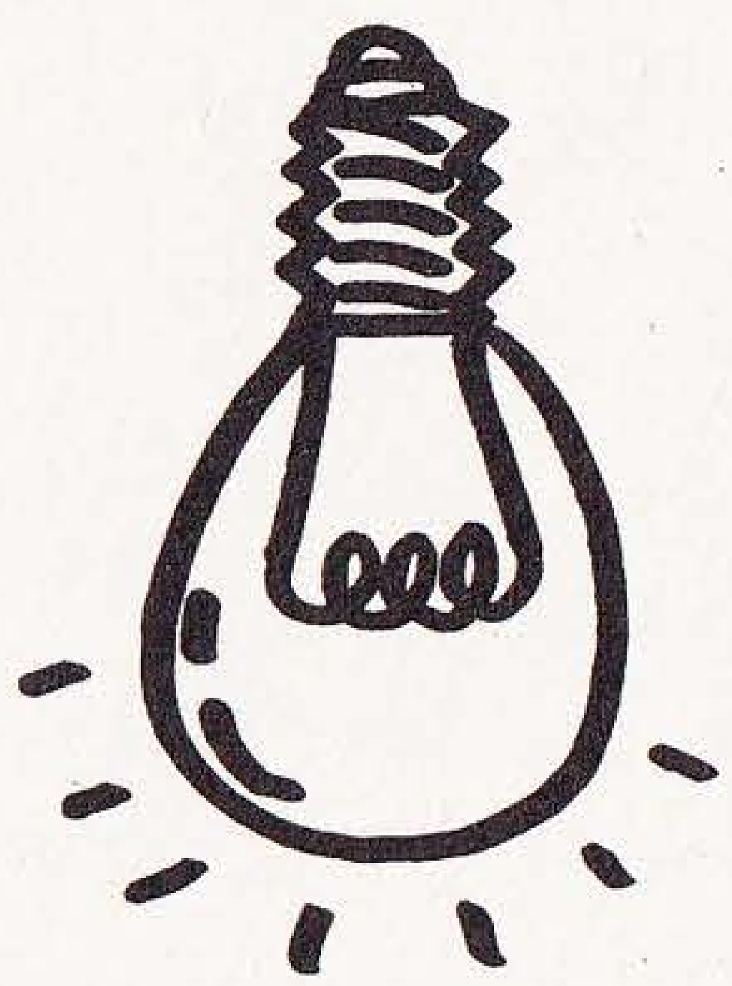


アプローチ3 全体の流れ図を書く

まずゼネラルフローチャートを書きます(**図 6**).

図 6





メインルーチンとは「プログラムの主要部分」という意味で，ゲーム実行中につねに働いている部分をさします．このメインルーチンは，衝突の判定のところに載せたフローチャートの部分に当たります．

なお，ただルーチンという場合は，プログラムの一部のことをさします．ルーチンという言葉は，今後しばしば登場しますから，よく覚えておきましょう．

3節 プログラムを組む

アプローチ1 主な変数を設定する

まず変数名を決めておきます．この程度の大きさのプログラムの場合には，とくに問題が起きないかもしれませんが，変数名をしっかりと決めておかないとプログラムを組んでいる際にわからなくなったり，別の用途に同じ変数を使ってしまったりということがよく起こります．そこで，ここでは**表1**のように主な変数を設定することにしました．

表1

変　数	内　容
X 1，Y 1	左側のプレイヤーの点の横座標，縦座標
X 2，Y 2	右側のプレイヤーの点の横座標，縦座標
U 1，V 1	左側のプレイヤーの点の横方向の動き，縦方向の動き
U 2，V 2	右側のプレイヤーの点の横方向の動き，縦方向の動き
B(i，j)	(i，j)の座標が壁なら1，そうでないなら0を値にもつ
L	左側のプレイヤーの勝った回数
R	右側のプレイヤーの勝った回数
T	引き分けの回数

アプローチ2 メインルーチンを組む

さて，まず**リスト1**を見てください．これが筆者が最初に組んでみたメインルーチンです．各行の役割を説明すると，次の**表2**のようになります．とくに難しい内容はないと思いますが，830行から870行に出てくるCHR＄関数について，少し

説明をしておきましょう.

リスト1

```
700 REM メイン ルーチン
710 B(X1,Y1)=1:B(X2,Y2)=1
720 LOCATE X1,Y1
730 PRINT "●";
740 LOCATE X2,Y2
750 PRINT "O";
780 A$=INKEY$
790 IF A$="w" THEN U1=0:V1=-1
800 IF A$="a" THEN U1=-1:V1=0
810 IF A$="s" THEN U1=1:V1=0
820 IF A$="z" THEN U1=0:V1=1
830 IF A$=CHR$(28) THEN U2=1:V2=0
840 IF A$=CHR$(29) THEN U2=-1:V2=0
850 IF A$=CHR$(30) THEN U2=0:V2=-1
860 IF A$=CHR$(31) THEN U2=0:V2=1
870 A$=INKEY$
880 IF A$<>"" THEN 790
890 X1=X1+U1:Y1=Y1+V1
900 X2=X2+U2:Y2=Y2+V2
920 IF B(X1,Y1) THEN 1010
930 IF B(X2,Y2) THEN 1030
940 GOTO 710
```

表 2

行番号	内　容
710	現在ある点を壁であるとみなす
720～730	左側のプレイヤーの点を画面に出す
740～750	右側のプレイヤーの点を画面に出す
780	キーから1文字読み，変数A$に代入
790～860	どのキーが押されたかにより進む向きを変える
870～880	他にもキーが押されていないか調べ，押されていれば790行へ行く
890～900	次の点の位置を求める
920～930	求めた点が壁かどうか調べ，壁ならばそれぞれの処理先へ飛ぶ
940	メインルーチンの先頭に戻る

コンピュータで使われる文字は，それぞれキャラクタコードと呼ばれる数字で表わすことができます．BASIC文法書の巻末などについているキャラクタコード表を見ればその対応はすぐにわかりますが，たとえばAという文字は65番，Bという文字は66番といった具合になっています．AとかBとかいう文字なら"A"，"B"という文字式で表わすことができるのですが，[↓]，[→]といったキーは"↓"のように表わせないので，CHR$関数を使います．各キーとコードとの対応は，**図7**のようになっています．ちなみに，各キーがどのようなキャラクタコードを持っているかを調べるには，**リスト2**のようなプログラムを作り，RUNさせてから各キーを押せばわかります．[INS]とか[RETURN]とかいったキーがどういうコードを持っているか，一度試してみましょう．このような""で囲んで表わせない文字(コントロール・キャラクタという)のうち，ゲームでよく使われるものは，さきほどの[→]，[←]，[↑]，[↓]と[RETURN](コードは13番)で，これらのコード番号は覚えておくと便利です．

図7

コード	対応するキー
28	[→]
29	[←]
30	[↑]
31	[↓]

リスト2 キャラクタコード確認プログラム

```
10 A$=INKEY$
20 IF A$="" THEN 10
30 PRINT ASC(A$)
40 GOTO 10
```

さて，せっかくメインルーチンを組んでも，その前の準備（壁を描いたり，配列変数に最初の値を設定したり）がなければ，羽のないトンボのようなもので，プログラムは動きません．

そこで，このような準備段階のプログラムを作ってみましょう．**リスト3**は画面の初期設定をするものです．これで，前節で用意したスクリーンレイアウトが

リスト3

```
130 WIDTH 32:KEY OFF
600 REM がめんの しょきせっていをする
610 COLOR 15,5:CLS
620 A$=" "+STRING$(30,"*")
630 B$=" *"+SPACE$(28)+"*"
640 PRINT A$
650 FOR I=1 TO 22:PRINT B$:NEXT
660 PRINT A$;
```

実際の画面上に実現されます．コマンドについて簡単な説明を加えると，

WIDTH　32……画面の横方向の文字数を32文字にする

KEY　OFF……画面下部のファンクションキーの内容表示を消す

STRING$(30,　"*")……*が30個つらなった文字列

SPACE$(28)……空白が28個つらなった文字列

というようになります．

次の**リスト4**は配列B(X，Y)の初期設定をするものです．210～220行で，外側の壁の部分を表わす配列の値を1にしています．310～340行は，FOR～NEXTループが2重になっているのでわかりにくいかもしれませんが，壁に囲まれた残り部分の配列の値をすべて0にするものです．

最後に，**リスト5**で点の最初の場所と方向を決めます．これはただの代入文ですから説明はいりませんね．

リスト4

```
120 DIM B(30,23)
200 REM しょうとつ はんていよう はいれつを つくる (そとがわ)
210 FOR I=1 TO30:B(I,0)=1:B(I,23)=1:NEXT
220 FOR I=1 TO22:B(1,I)=1:B(30,I)=1:NEXT
300 REM しょうとつ はんていよう はいれつを つくる (うちがわ)
310 FOR I=2 TO 29
320 FOR J=1 TO 22
330 B(I,J)=0
340 NEXT J,I
```

リスト5

```
350 REM テンの いちの しょきせってい
360 X1=4:Y1=3:X2=27:Y2=20
370 U1=1:V1=0:U2=-1:V2=0
```

さて，以上でメインルーチンの手足となる最低限のものはできたわけです．リスト1の920，930行で勝敗が決したときの処理先が未定義のままでしたから，とりあえず，

```
1010 END
1030 END
```

とでも追加してみましょう．また，このプログラムで使う数値変数には，小数とか分数はでてこないので，すべて整数型ですみます．そこで，

```
100 REM しょきせってい する
110 DEFINT A-Z
```

の2行を追加します．整数型のほうが計算速度は速いので，速さを求められるゲームでは，整数しか扱わない変数はすべて整数型に指定しておくことが大事です．

以上のプログラムをすべてまとめて，動かすとどうなるか？――興味のある人は自分で試してほしいと思いますが，ここでは結果を報告することにします．

①そのままでは点の動きが速すぎた．

②まれにおきるケースではあるが，同一地点に2人のプレイヤーの点が重なると，衝突を起こさずに通り抜けてしまう．

このような問題点が生じてしまいました．そこで①に対する解決策としては

```
770 FOR I=1 TO 99:NEXT
```

という1行を入れることにしました．この文は時間かせぎをするだけで，なんの仕事をするわけでもありません．このような文を入れることを「ウェイトを入れる」と言います．ゲームにおいては，適当な場所でウェイトを入れて速度を調節す

ることも大事です．

次に②に対する解決策として，

```
910 IF X1=X2 AND Y1=Y2 THEN C=3:GOTO1040
1140 END
```

という2行を入れることにしました．910～930の3行で，ようやく壁に対する衝突のチェックが完璧になったわけです．

以上で，バリケードゲームの大筋は完成です．まだ音もないし，勝敗の表示もない状態ですが，それは次の段階で考えることにしましょう．現在のプログラム状況を**リスト6**にあげます．

リスト6

```
100 REM しょきせってい する
110 DEFINT A-Z
120 DIM B(30,23)
130 WIDTH 32:KEY OFF
200 REM しょうとつ はんていよう はいれつを つくる (そとがわ)
210 FOR I=1 TO 30:B(I,0)=1:B(I,23)=1:NEXT
220 FOR I=1 TO 22:B(1,I)=1:B(30,I)=1:NEXT
300 REM しょうとつ はんていよう はいれつを つくる (うちがわ)
310 FOR I=2 TO 29
320 FOR J=1 TO 22
330 B(I,J)=0
340 NEXT J,I
350 REM テンの いちの しょきせってい
360 X1=4:Y1=3:X2=27:Y2=20
370 U1=1:V1=0:U2=-1:V2=0
600 REM がめんの しょきせっていをする
610 COLOR 15,5:CLS
620 A$=" "+STRING$(30,"*")
630 B$=" *"+SPACE$(28)+"*"
640 PRINT A$
650 FOR I=1 TO 22:PRINT B$:NEXT
660 PRINT A$;
700 REM メイン ルーチン
710 B(X1,Y1)=1:B(X2,Y2)=1
720 LOCATE X1,Y1
730 PRINT"●";
```

```
740 LOCATE X2,Y2
750 PRINT"O";
770 FOR I=1 TO 99:NEXT
780 A$=INKEY$
790 IF A$="w" THEN U1=0:V1=-1
800 IF A$="a" THEN U1=-1:V1=0
810 IF A$="s" THEN U1=1:V1=0
820 IF A$="z" THEN U1=0:V1=1
830 IF A$=CHR$(28) THEN U2=1:V2=0
840 IF A$=CHR$(29) THEN U2=-1:V2=0
850 IF A$=CHR$(30) THEN U2=0:V2=-1
860 IF A$=CHR$(31) THEN U2=0:V2=1
870 A$=INKEY$
880 IF A$<>"" THEN 790
890 X1=X1+U1:Y1=Y1+V1
900 X2=X2+U2:Y2=Y2+V2
910 IF X1=X2 AND Y1=Y2 THEN C=3:GOTO 1040
920 IF B(X1,Y1) THEN 1010
930 IF B(X2,Y2) THEN 1030
940 GOTO 710
1010 END
1030 END
1140 END
```

アプローチ3 改良を加える

この項ではアプローチ2で作ったプログラムをもとにさらに改良を加えてみましょう.

1．音を出す

点が進んでいる間は,ドッドッドッという音がほしいと思います．そこで,

```
140 PLAY "S0M3000T240O2"
760 PLAY "C32"
```

の2行を追加します．140行は音の出し方，長さ，音階を決めている行で，実際に音を出すのは760行のほうです.

2．ゲームの始まりかたを変える

アプローチ2のプログラムだけでは，いきなりゲームがはじまる感じです．そこで，プレイヤーの許しを得てからはじまるように変えてみます.

```
500 CLS:LOCATE 6,12
510 PRINT"はじめて いいかな ?  [RET]"
520 A$=INKEY$
530 IF A$<>CHR$(13) THEN 520
```

上の4行を追加します．すると，500行で画面を消し，510行で画面にメッセージを出したあと，RETURNを押さないかぎり，520～530行の間をぐるぐると回ります．つまりRETURNを押すのがゲームスタートの合図となるのです．

3．勝敗の表示をする

4．何回でも繰り返して遊べるようにする

ここでつけ加えようとするのは，

ⓐその回の勝敗を表示する

ⓑ通算での勝敗数を表示する

ⓒ何回でも繰り返せるようにする

の3点です．まず，勝敗がついたかどうかは，

```
910 IF X1=X2 AND Y1=Y2 THEN C=3:GOTO1040
920 IF B(X1,Y1) THEN 1010
930 IF B(X2,Y2) THEN 1030
```

の部分でわかるのですが，単純に考えれば，1040行に飛ぶ場合は引き分け，1010行に飛ぶ場合は2番のプレイヤー（つまり右のプレイヤー）の勝ち，1030行に飛ぶ場合は左のプレイヤーの勝ち，となりそうです．しかし，B(X1，Y1)が0でないからといって，すぐに右のプレイヤーの勝ちになるかというと，同時にB(X2，Y2)も0でないかもしれないので注意が必要です．以上を考えに入れて勝敗の表示ルーチンとして作ったのが，**リスト7**です．

リスト7

```
910 IF X1=X2 AND Y1=Y2 THEN C=3:GOTO1040
920 IF B(X1,Y1) THEN 1010
930 IF B(X2,Y2) THEN 1030
940 GOTO 710
1000 REM しゅうりょうご"の しょり ルーチン
1010 IF B(X2,Y2) THEN C=3 ELSE C=2
```

```
1020 GOTO 1040
1030 C=1
1040 FOR I=11 TO 13
1050 LOCATE 9,I
1060 PRINT SPACE$(13)
1070 NEXT
1080 LOCATE 10,12
1090 ON C GOTO 1100,1120,1140
1100 PRINT"ひだりの ひとの かち":L=L+1
1110 GOTO 1500
1120 PRINT"みぎの ひとの かち":R=R+1
1130 GOTO 1500
1140 PRINT"  ひ き わ け":T=T+1
1500 REM つうさんの しょうはいの ひょうじ
1510 FOR I=1 TO 3000:NEXT
1520 LOCATE 10,11:PRINT"みぎ—";R;"しょう"
1530 LOCATE 10,12:PRINT"ひだりー";L;"しょう "
1540 LOCATE 10,13:PRINT"ひきわけー";T;"かい"
1550 GOTO 300
```

このリストのなかに出てくるCという変数は，勝ち負けを表わしています．

C＝1 ──→左のプレイヤーの勝ち

C＝2 ──→右のプレイヤーの勝ち

C＝3 ──→ひきわけ

というわけです．

1040～1070行で行なっているのは，勝敗を表示する部分の下地として，空白を書くという作業です．また1510行ではウェイトを入れ，ある程度の時間，対戦成績の表示をしているようにしています．そして最後の，

```
1550 GOTO 300
```

という文ですが，これによって，ふたたびゲーム開始になるわけです．300行へ飛ぶ理由は，壁に囲まれた部分の配列の内容を，もとの状態に戻す必要があるからです．

5．その他

以上で改良は終わりにしますが，皆さんがさらに手を加えてくださることを期

待します．改良のヒントとしては，

① 勝敗がついた時点で音楽を流す

② はじまる点の位置を一定にしないで，乱数などを使って変える

などが考えられます．また，アイデアしだいで，このゲームにいろいろなバリエーションを持たせることもできるでしょう．たとえば，

①「アクセルキー」というのを設ける．アクセルキーを押すと点の進む速さが倍になる．

②「ぶつかったら負け」というゲームではなく，「いかに大きな陣地をとるか」というゲームにする．外側の壁や相手の作った壁にぶつかっても，ゲームが終了するのではなく，ただその方向に進めないというだけにする．

などといったことが考えられます．

最後に外側の壁の表示法を変える方法を載せておきましょう．キャラクタコード255番の文字が，じつはカーソルがある位置と同じキャラクタを持つことを利用します．620，630行を次のように変更します．

```
620 A$="  "+STRING$(30,255)
630 B$=A$:MID$(B$,3,28)=SPACE$(28)
```

もとのプログラムと形は違いますが，結局"＊"のかわりにCHR＄(255)を置いただけのことです．

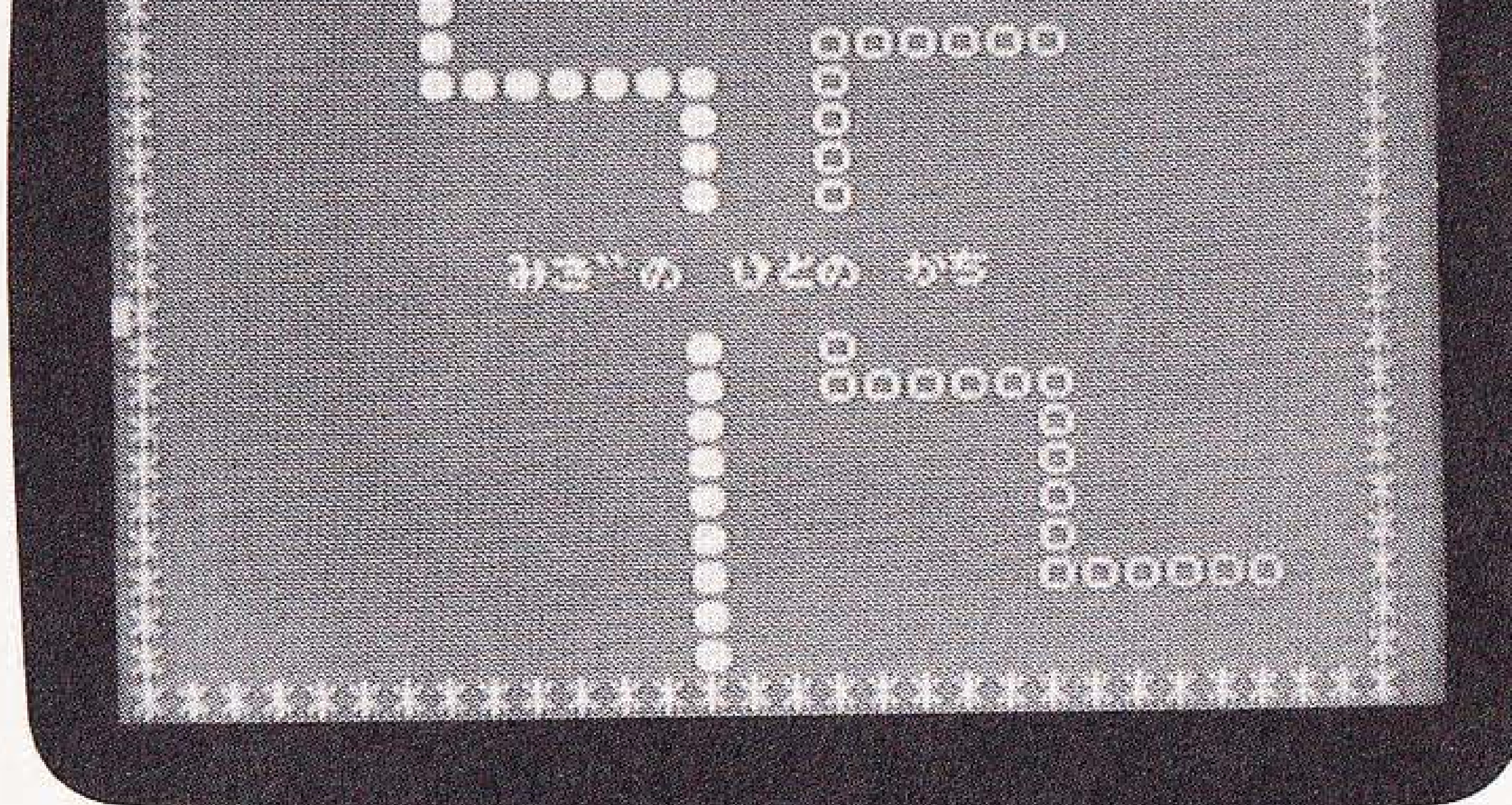
このプログラムを実行しているときに，STOPを押して一時停止させてみましょう．はたして画面はどうなるか，やってみてのお楽しみ！

リスト8　バリケードゲーム全リスト

```
10 REM
20 REM  Barricade Game
30 REM
40 REM      ver 1.30
50 REM
100 REM しょきせってい する
110 DEFINT A-Z
120 DIM B(30,23)
130 WIDTH 32:KEY OFF        ←ファンクションキーの表示を停止
140 PLAY"S0M3000T24002"
```

```
200 REM しょうとつ はんていよう はいれつを つくる (そとか゛わ)
210 FOR I=1 TO 30:B(I,0)=1:B(I,23)=1:NEXT  } 壁の部分は
220 FOR I=1 TO 22:B(1,I)=1:B(30,I)=1:NEXT  } 1にする
300 REM しょうとつ はんていよう はいれつを つくる (うちか゛わ)
310 FOR I=2 TO 29  }
320 FOR J=1 TO 22  } 壁の内側を0にする
330 B(I,J)=0       }
340 NEXT J,I       }
350 REM テンの いちの しょきせってい
360 X1=4:Y1=3:X2=27:Y2=20  } 初めの位置と方向を決める
370 U1=1:V1=0:U2=-1:V2=0   }
500 CLS:LOCATE 6,12
510 PRINT"はし゛めて いいかな ?  [RET]"  } 始めの合図に
520 A$=INKEY$                           } RETURN を求める
530 IF A$<>CHR$(13) THEN 520            }
600 REM か゛めんの しょきせっていをする
610 COLOR 15,5:CLS
620 A$=" "+STRING$(30,"*")
630 B$=" *"+SPACE$(28)+"*"
640 PRINT A$  ←上の壁を描く
650 FOR I=1 TO 22:PRINT B$:NEXT  ←左右の壁を描く
660 PRINT A$; ←下の壁を描く
700 REM メイン ルーチン
710 B(X1,Y1)=1:B(X2,Y2)=1  ←今いる点を以降は壁とみなす
720 LOCATE X1,Y1  }
730 PRINT"●";     } 新しい点を打つ
740 LOCATE X2,Y2  }
750 PRINT"○";     }
760 PLAY "C32"
770 FOR I=1 TO 99:NEXT  ←ウェイトを入れる
780 A$=INKEY$
790 IF A$="w" THEN U1=0:V1=-1
800 IF A$="a" THEN U1=-1:V1=0                  }
810 IF A$="s" THEN U1=1:V1=0                   }
820 IF A$="z" THEN U1=0:V1=1                   }
830 IF A$=CHR$(28) THEN U2=1:V2=0    [→]       } 進む方向を変える
840 IF A$=CHR$(29) THEN U2=-1:V2=0   [←]       }
850 IF A$=CHR$(30) THEN U2=0:V2=-1   [↑]       }
860 IF A$=CHR$(31) THEN U2=0:V2=1    [↓]       }
870 A$=INKEY$
880 IF A$<>"" THEN 790
890 X1=X1+U1:Y1=Y1+V1  } 次の点の位置を求める
900 X2=X2+U2:Y2=Y2+V2  }
910 IF X1=X2 AND Y1=Y2 THEN C=3:GOTO 1040←出会いがしら
                                          の衝突を調べる
```

```
920 IF B(X1,Y1) THEN 1010   } 壁かどうか調べる
930 IF B(X2,Y2) THEN 1030   }
940 GOTO 710
1000 REM しゅうりょうご゛の しょり ルーチン
1010 IF B(X2,Y2) THEN C=3 ELSE C=2
1020 GOTO 1040
1030 C=1
1040 FOR I=11 TO 13         }
1050 LOCATE 9,I             } 画面の中央にメッセージ表示窓を作る
1060 PRINT SPACE$(13)       }
1070 NEXT                   }
1080 LOCATE 10,12
1090 ON C GOTO 1100,1120,1140
1100 PRINT"ひた゛りの ひとの かち":L=L+1
1110 GOTO 1500
1120 PRINT"みぎ゛の ひとの かち":R=R+1
1130 GOTO 1500
1140 PRINT"  ひ き わ け":T=T+1
1500 REM つうさんの しょうはいの ひょうじ゛
1510 FOR I=1 TO 3000:NEXT
1520 LOCATE 10,11:PRINT"みぎ゛——";R;"しょう"
1530 LOCATE 10,12:PRINT"ひた゛りー";L;"しょう "
1540 LOCATE 10,13:PRINT"ひきわけー";T;"かい"
1550 GOTO 300
```



3

（実践編 II）

南極米・ソ戦ゲーム



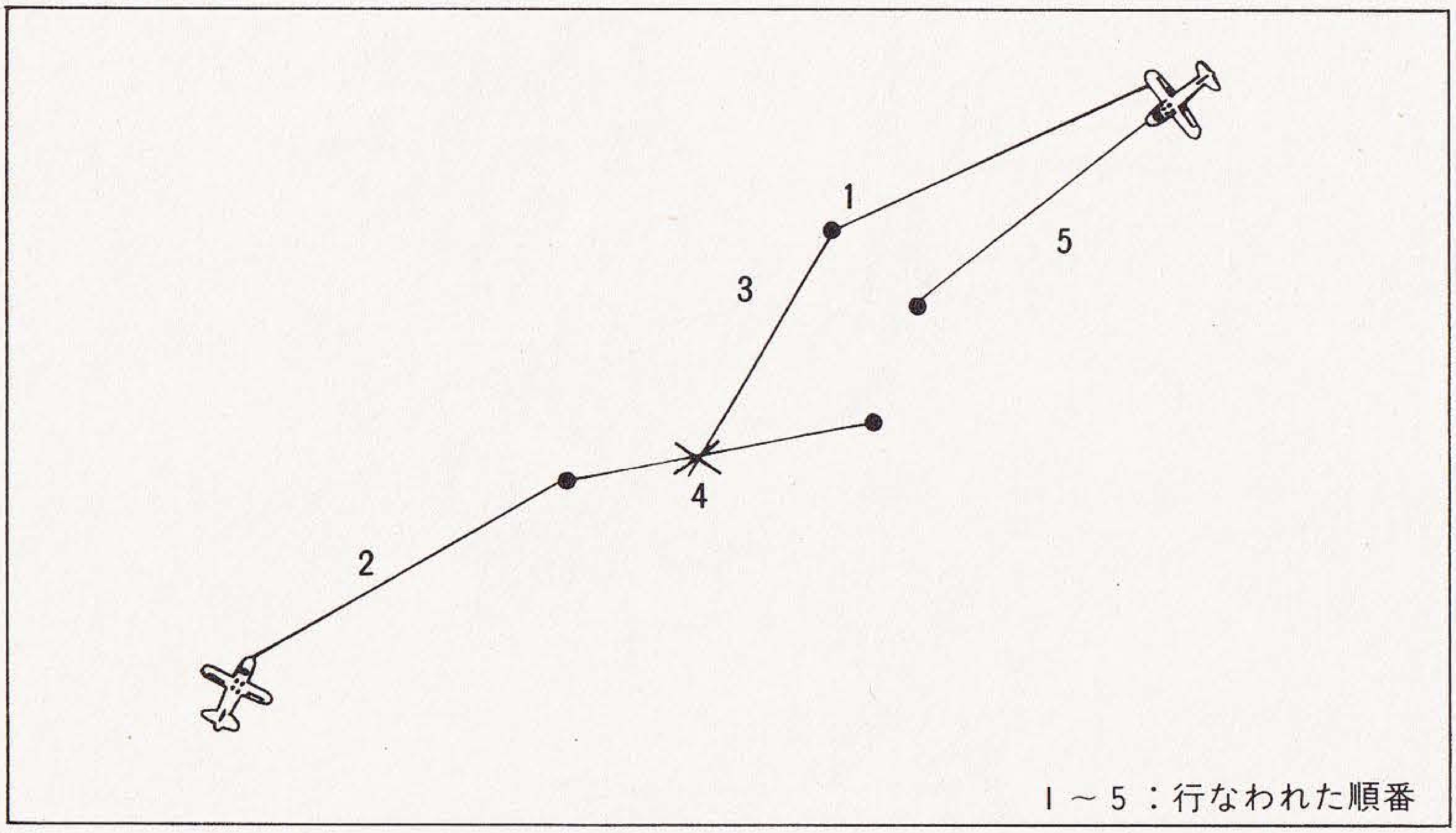
1節　ゲームの紹介

米・ソ戦ゲームなどというと，ウォーゲームかと思って喜ぶ人，逆に戦争ゲームなんてきらいだと言う人，といろいろな反応があるかもしれません．しかしこれはいわゆるウォーゲームではありません．

「鉛筆倒し」という遊びを知っていますか．２人以上の人数で行なうゲームで，１枚の紙の上に，各人が「陣地」を作ります．筆者の地方では漢字の「士」の字を書いてそれを「陣地」がわりにしていました．

次にこの「陣地」を基点として，紙の上に鉛筆を斜めに立てて，その鉛筆を倒します．そうすると紙の上には直線状の鉛筆の跡が残ります．この動作を「たまを飛ばす」といって，鉛筆の跡の終点(かすれて見えなくなるところ)に「・」印を書いて，そこまでたまが飛んだとみなすのです．

鉛筆倒し実戦図



この「たま飛ばし」を各人が順番に行ないます．そして２回目からは陣地からたまを飛ばさなくても，前の「・」印のところからたまを飛ばしてもよいのです．

「鉛筆倒し」の最終目的は，相手の陣地を破壊することにあります．３人以上の場合なら，最後の１人の勝者が決まるまでゲームを行ないます．それでは，どう

したら「陣地を破壊」できるかというと，相手の陣地のマーク(士の字)を「たま」で貫けばよいのです．同様に相手の「たま」を破壊することもできます．この場合は，相手のたまのあるところ(「・」のあるところ)を自分のたまが貫けばよいのです．

さて，「鉛筆倒し」の説明を書いてきましたが，これは「南極米・ソ戦ゲーム」というのが，じつはこの「鉛筆倒し」とほぼ同じだからです．

<南極の大雪原に米・ソ両国の基地があった．たがいに極地においては協力し合っていた両国であったが，西暦20××年両国は交戦状態にはいった．そして，この南極基地でも，相手基地に向けてICBM(大陸間弾道弾)のボタンが押された…….>

一応このようなストーリーを想定し，「南極米・ソ戦ゲーム」と名づけたわけです．しかしゲームのタイトルやストーリーなどは，この際どうでもよいことです．相手の小さな「たま」に自分の「たま」の弾道を合わせて発射する，あの微妙な角度と強さの加減の難しさを，マイコンの上に再現してみようと思い，ここに取り上げてみたわけです．

そのようなわけで，他のゲーム(バリケードとブロックくずし)がアーケードゲームをそのまま取り入れたものなのに対し，この本の中では唯一のオリジナルゲーム(アイデアを既成のマイコンゲームから借りたのではないもの)になっています．

2節 ゲームの骨格を作る

さて，まずはゲームの構想をもう少し固めたいのですが，文章で述べるより，図の形で頭に入れたほうがわかりやすいと思うので，まずはスクリーンレイアウトを載せることにしましょう．実際のゲーム作りの過程においては，このレイアウトはあとで述べる構想をもとに作りました．

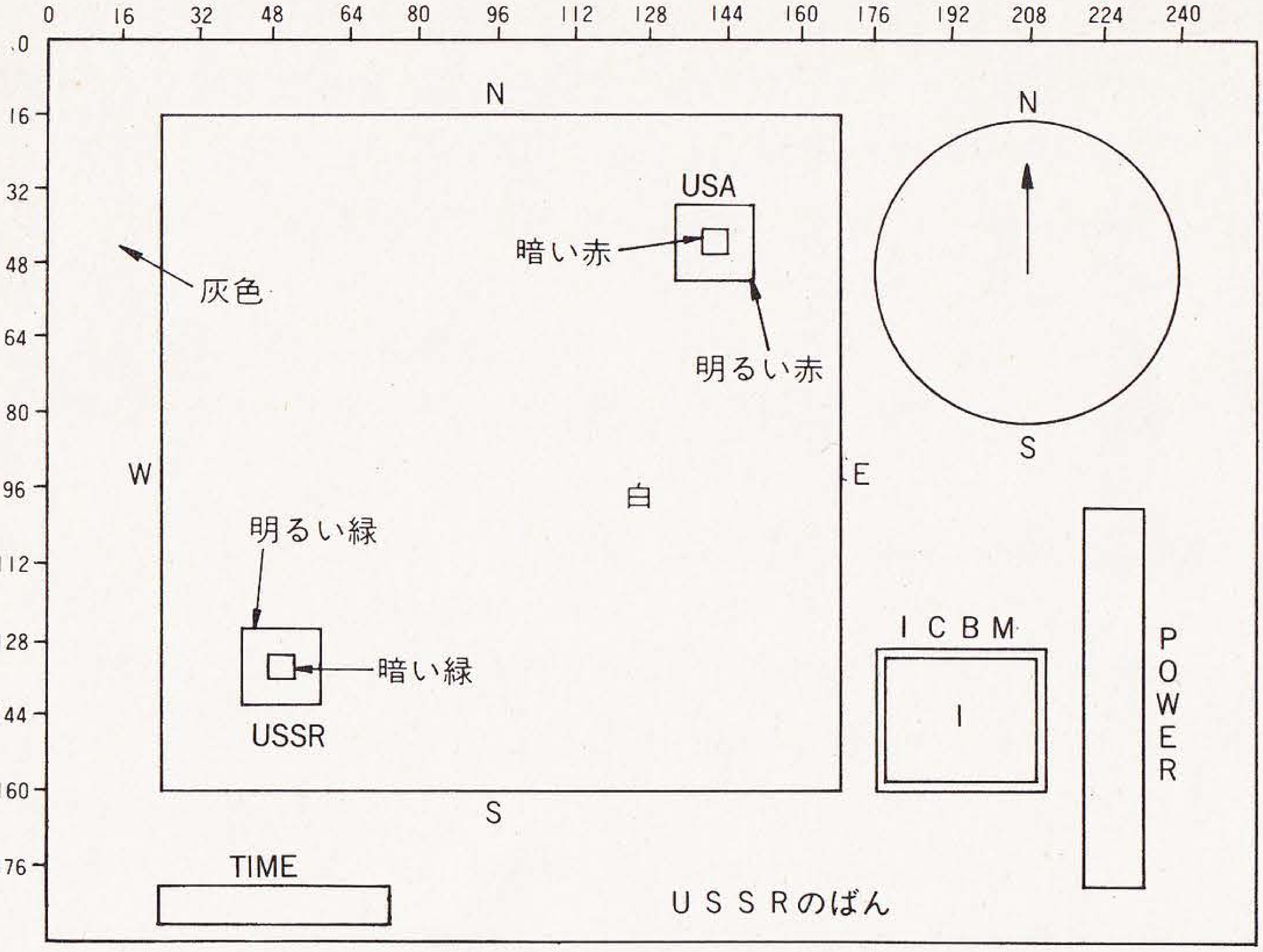
アプローチ1 レイアウトを決める

レイアウトの説明をしましょう．

まずスクリーンモードですが，たくさんの色を使い，細かいものも描くので，**モード2**しかないということになりました．

画面の大部分を占める四角のエリアが戦場となる部分で，南極ですからやはり

南極米・ソ戦スクリーンレイアウト



色は白です．右上にある円はICBMの飛ぶ向きを示す方向計，右下のPOWERと書いてある枠がたまの落下地点までの距離計です．この2つをセットすることにより，ICBMの飛び方をコントロールするわけです．

距離計の左にある小さな枠はICBMの番号を示す欄です．そしてその下に「USSRのばん」とありますが，ここに米・ソどちらの攻撃であるかを表示します．ちなみに，USSRはソ連の略称で，USAはアメリカです．

ゲームに緊迫感を盛り込むために攻撃時間に制限を設けました．その時間を示すのが，画面左下の表示計(TIME)です．制限時間を越えると自動的に相手の攻撃になります．

その他に，画面上にはN, S, W, Eという文字がありますが，これはそれぞれ，北，南，西，東という方位を表わしています(もしも画面の中心が南極点なら，全部が北ということになりますね)．

最後に色について…．戦場は白，背景は灰色と決めましたが，その他の部分は実際に動かしてみて，どの色がよいか決めることにしました．一般に，色の選択

は，プログラムを作ってからあれこれ試して決めることが多いのです．

アプローチ2 メーター類について

アプローチ１の段階ですでに「構想」の多くがはいっています．距離計，方向計，時間計の設置といったことがそれです．ここでは，まずこれらのメーター類の使い方について述べましょう．

1．ICBM番号の表示欄

鉛筆倒しには，たまの数に制限はないのですが，南極米・ソ戦においてはたまは4つまでということにしました．理由は，このほうが作戦を練る楽しみが出てくると思ったからです(たとえばICBMの無駄使いができない)．適当な制限を設けておくことは，ゲームのおもしろさを増す上で大事なことです．

この欄は，4つのICBMのうちどれを動かすかを決めるためのものです．自分の番になったらまず動かしたいICBMの番号(１～４)をキーから入力し，さらに確認のために RETURN を押して選択します．そのとき入力したICBMの番号がこの窓に表示されます．

2．方向計

ICBMの番号の指定を行なったら，次はその方向を決めます．方向計の中の針が自分の進みたい方向をさしたときに RETURN を押せば，その方向にICBMは飛びます．

最初の構想では，針がランダム(でたらめ)に方向を指すようにして，それがたまたま自分の進ませたい方向を向いた瞬間に RETURN を押すようにしました．ところが，実際にプログラムを作ってゲームをしてみると，あまりにも難しいのでやめにしました．そして最終的には時計の針のように順に回っていく仕組みになったのです．

3．距離計

メーターの中を赤い棒(エネルギー充てん度を示す)が昇っては降りするので，適当なところで RETURN を押します．するとその時点でメーターが止まり，ICBMの飛ぶ距離が決定されます．

4．時間計

動きは距離計と同じですが，こちらはだんだんと攻撃時間が減っていく様子を示します．

時間計



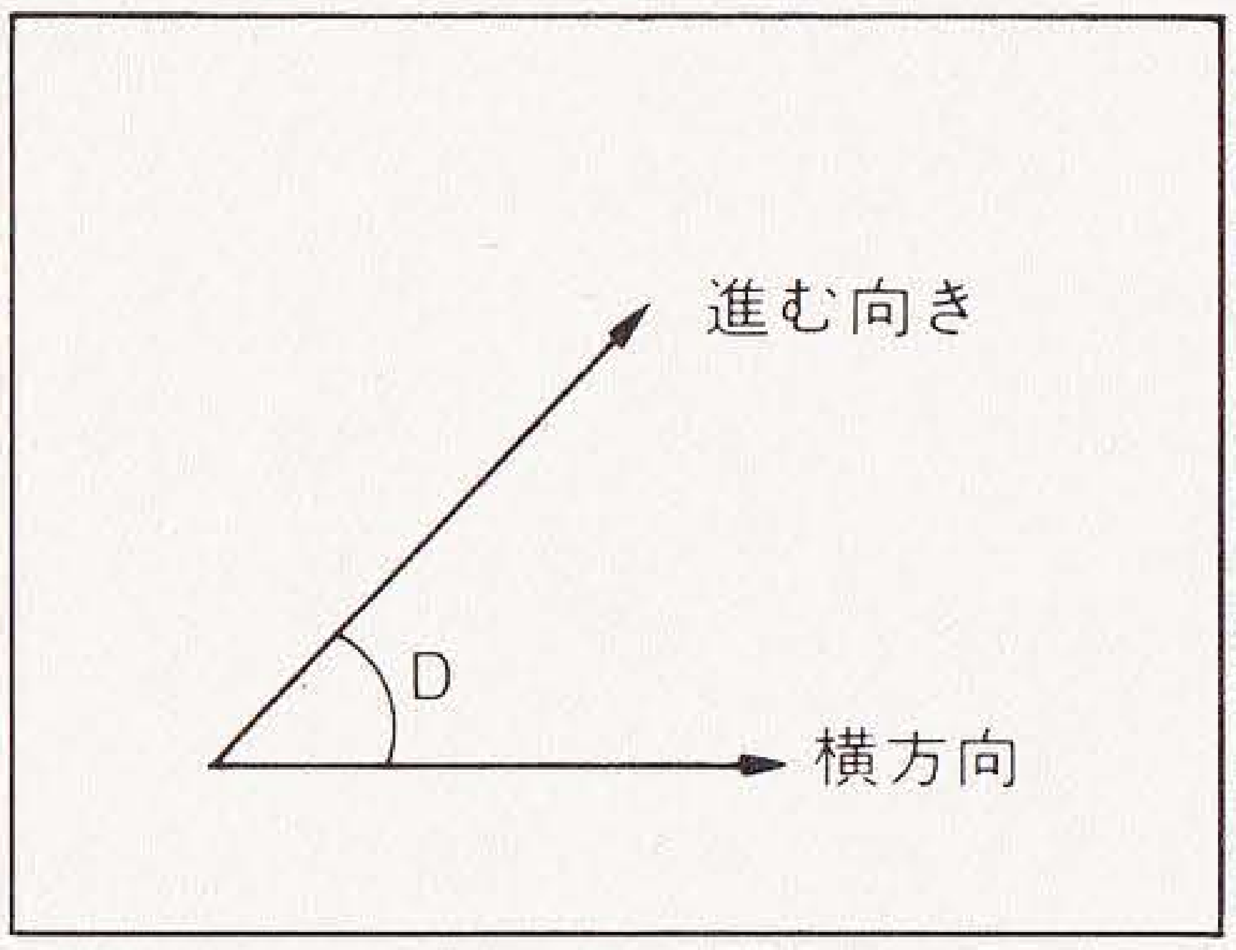
だんだん黄色の部分が減る

アプローチ3 たまの動かし方

1．たまを動かす論理

たまの番号，方向，距離をセットすると，次はたまを動かす番です．ここでたまの番号をI，方向をD，距離をPで表わし，I番のたまのある場所を(X(I),Y(I))としてみましょう．方向Dは，**図8**のように横方向(X座標が大きくなる向き)とたまの進む向きとがなす角とします．

図8



すると最終的にたまが到達する地点の座標は，次のように表わされます．

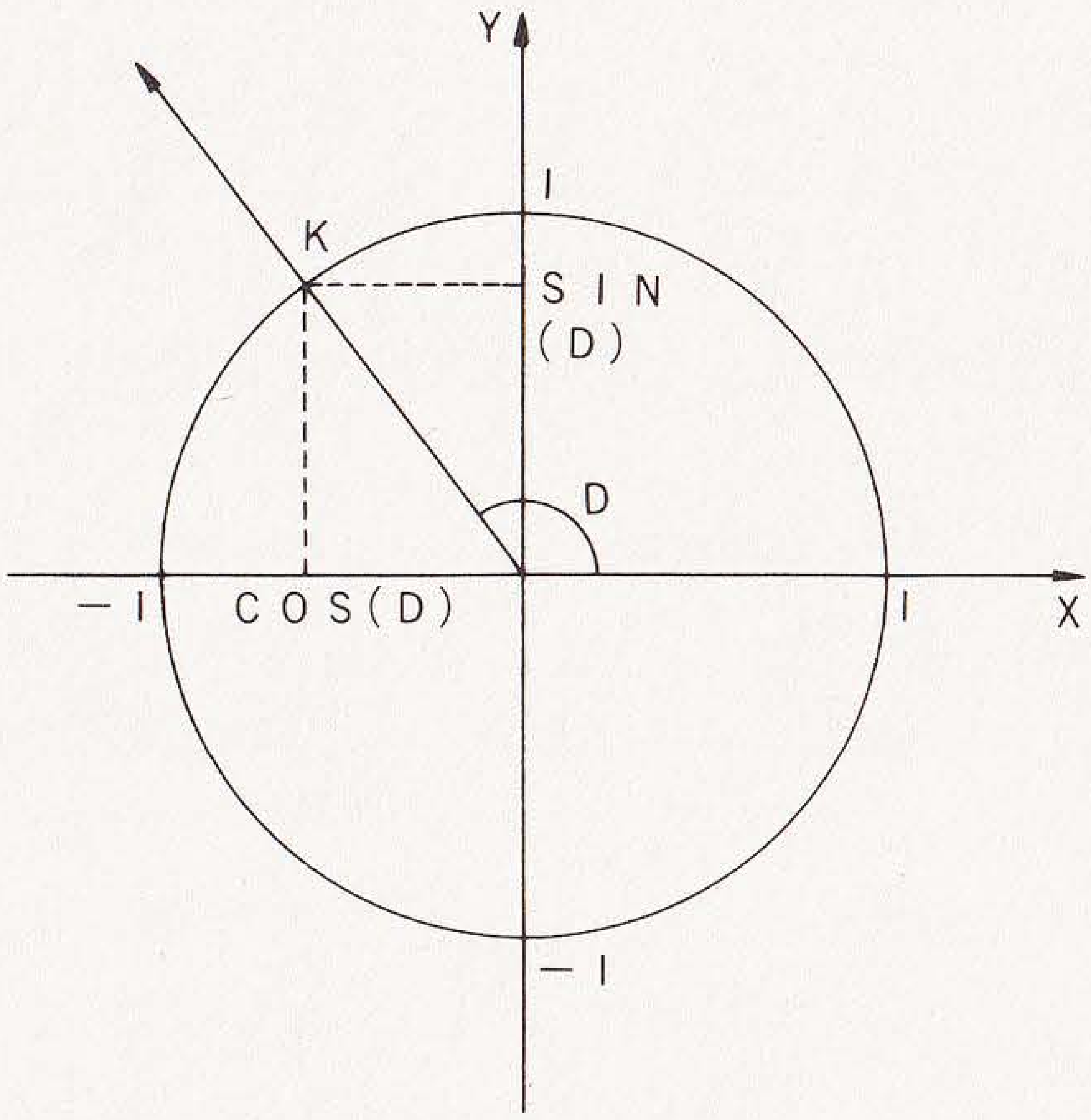
(X(I)＋P＊COS(D)，Y(I)＋P＊SIN(D))

ここでCOS, SINという関数が出てきましたが，一体何を計算しているのかわからない方も多いでしょう．**図9**を見てください．この円は半径が1の円です．X軸と角度がDある直線と，この円との交わる点をKとすると，このときのX, Y座標が，COS(D)，SIN(D)なのです．したがってもとの点からの長さがPで，角度がDの点なら，もとの点に比べてX，Yの方向に，

(P＊COS(D)，P＊SIN(D))

だけ離れた点となるわけです.

図9



さて到着する点の座標はわかったのですが，これだけでは途中で敵の基地やたまと衝突したかどうかを調べるには不便です．そこで，少しずつ動かすことを考えてみましょう.

```
10 FOR I=1 TO P
20 X(I)=X(I)+COS(D)
30 Y(I)=Y(I)+SIN(D)
40 NEXT I
```

それには上のようなプログラムにすればよいでしょう．つまり長さ1ずつをP回加えることによって，だんだんと動かすわけです．そして実際には，このFOR～NEXTの間に，敵の破壊の有無や，戦場外に飛び出してしまったかどうかなどの判定プログラムを入れてやればよいわけです.

2．衝突の判定など

さて，たまが戦場外に出てしまったらどうするのか？　また，たまが命中したときにはどうするのか？　このあたりをはっきりさせないまま，ここまできました．そこで，それらをはっきりと決めておきましょう.

(a)範囲外へ出たときの取り扱い

ICBMについての自然な(?)考えからいけば，範囲外に出てしまったICBMはレーダーによる補捉ができなくなったものとして，それ以降使えないのが当然です．しかし，このゲームでは，範囲外に出てしまったICBMは自分の基地から再発射できるということにしました．敵陣深く攻めいったICBMが，自国の基地が危険なときには，わざと範囲外に出ることで防御にまわれるようにしたわけです．

(b)敵基地に命中した時

とにかく敵基地にICBMが当たれば，当てたほうの勝ちとします．

(c)ICBMに命中したとき

敵国のICBMにぶつかったときには，敵国のICBMを破壊するものとします．論理的に考えればこちらのICBMも同時に壊れて当然なのですが，その点は無視し，自分のほうはそのまま進めるものとします．いったん破壊されたICBMはそれ以降は使えず，したがって4発のICBMが破壊されたら自動的に敗けとなります．

また自国のICBMとの衝突はないものとします．自国のICBMどうしは互いに識別信号を出し合っているので，接近時には緊急回避行動がとられるんですねぇ．

3．スプライトの使い方

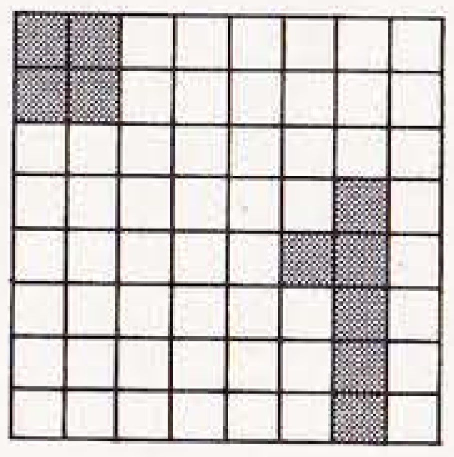
さらに，たまの表示法について考えてみます．たまはどんどん動いていくわけで，普通のグラフィックス画面に描くとすると，消したり描いたりがたいへんです．

そこでスプライトを使ってたまを表示することにしました．スプライトはいったん動かすものの形を定義してしまえば，あとで動かすのが非常に楽なので，このような場合には便利です．

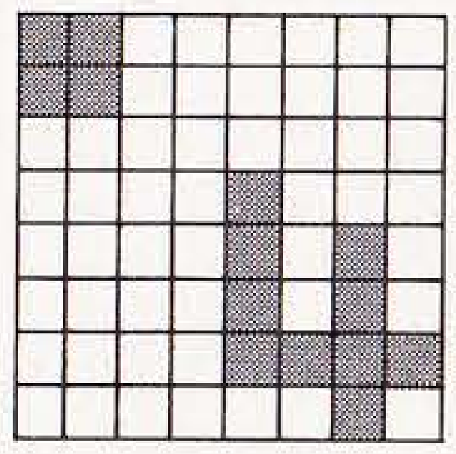
そこで**図10**のように，点とICBMの番号をまとめて1つのスプライトに定義することにしました．5番目にある「※」印のパターンは，あとで出てくるので，ここでは無視してください．スプライトの形(スプライト・パターン)の横に書いてある&B，&Hのついている数字は，それぞれ2進数，16進数です．2進数の0，1の並びのパターンは，それぞれスプライト・パターンの1つ1つの点に対応しているわけです．そこでたとえば，1つのたまのパターンをSPRITE$(1)に定義する場合，次のようなプログラムを組めばよいわけです．

```
10 A$=""
20 FOR I=1 TO 8
30 READ B$
40 A$=A$+CHR$(VAL("&H"+B$))
50 NEXT I
60 SPRITE$(1)=A$
70 DATA C0,C0,00,02,06,02,02,02
```

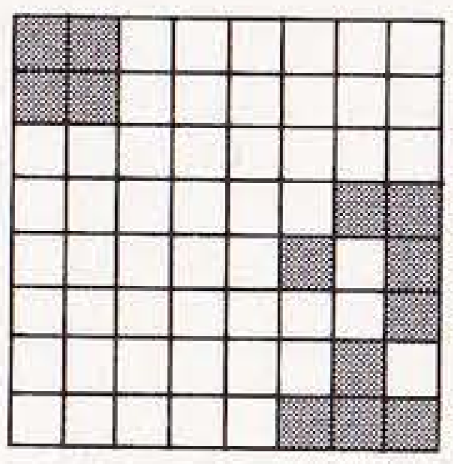
図10



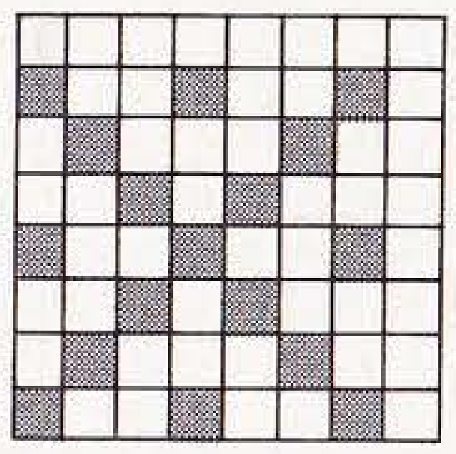
```
&B11000000=&HC0
&B11000000=&HC0
&B00000000=&H00
&B00000010=&H02
&B00000110=&H06
&B00000010=&H02
&B00000010=&H02
&B00000010=&H02
```



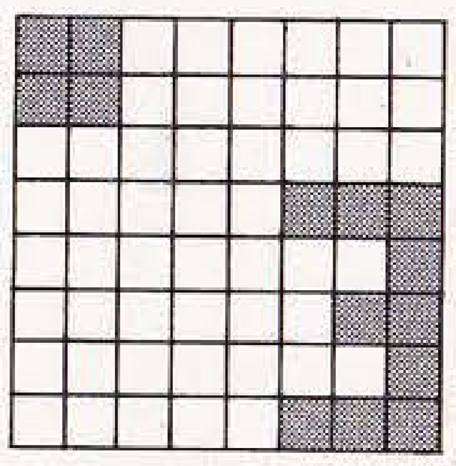
```
&B11000000=&HC0
&B11000000=&HC0
&B00000000=&H00
&B00001000=&H08
&B00001010=&H0A
&B00001010=&H0A
&B00001111=&H0F
&B00000010=&H02
```



```
&B11000000=&HC0
&B11000000=&HC0
&B00000000=&H00
&B00000011=&H03
&B00000101=&H05
&B00000001=&H01
&B00000010=&H02
&B00000111=&H07
```



```
&B00000000=&H00
&B10010010=&H92
&B01000100=&H44
&B00101000=&H28
&B10010010=&H92
&B00101000=&H28
&B01000100=&H44
&B10010010=&H92
```



```
&B11000000=&HC0
&B11000000=&HC0
&B00000000=&H00
&B00000111=&H07
&B00000001=&H01
&B00000011=&H03
&B00000001=&H01
&B00000111=&H07
```

アプローチ4 主要部分の論理を考える

すでにICBMを動かす論理などについては述べましたが，その他のやや難しいと思われる部分について考えてみましょう．

1．攻撃サイドが交代することをどうあらわすか？

かりに，USAの基地が(700，50)に，USSRの基地が(40，750)の位置にあったとします．USSRのICBMが(X，Y)の位置にあったとすると，そのICBMがUSAの基地と重なっていることの条件は，

```
10 BX=700:BY=50
20 IF X><BX THEN 60
30 IF Y><BY THEN 60
```

```
40 REM かさなっている
50 END
60 REM はず゛れている
```

と表わすことができます．しかし同じプログラムで，USAのICBMがUSSRの基地と重なっているかどうかの判定はできません．これに対処する方法として，次の3つの方法が考えられます．

①ほとんど同じルーチンを2つつなげる．

②片方の番が終わると変数を入れ換える．

```
10 BX1=700:BY1=50
20 BX2=40 :BY2=750
30 IF X><BX1 THEN 100
40 IF Y><BY1 THEN 100
50 REM めいちゅう
   ⋮
100 REM はず゛れ
    ⋮
200 REM へんすうのこうたい
210 SWAP BX1,BX2
220 SWAP BY1,BY2
230 GOTO 30
```

③配列を使い，配列の添字の値で区別する．

たとえば，USSRは0で表わし，USAは1で表わすとすると，前のところであげた例は，次のようになります．

```
10 BX(0)=700:BY(0)=50
20 BX(1)=40 :BY(1)=750
30 S=0
40 IF X><BX(S) THEN 100
50 IF Y><BY(S) THEN 100
60 REM めいちゅう
   ⋮
100 REM はず゛れ
    ⋮
```

```
        :
200 REM へんすうのこうたい
210 S=1-S
220 GOTO 40
```

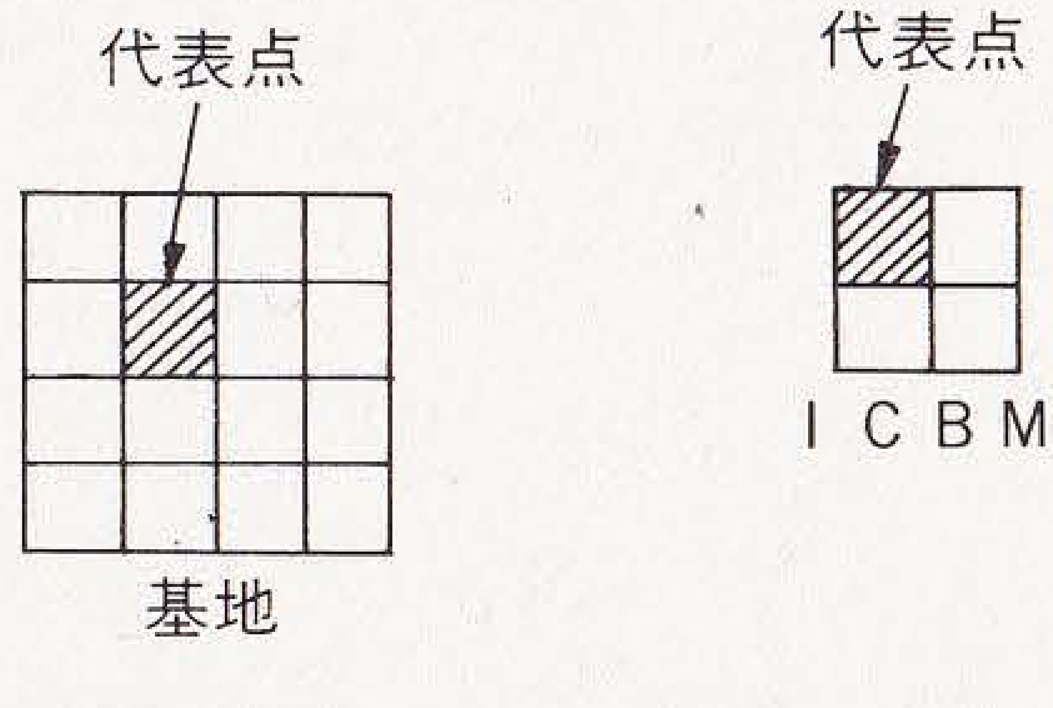
さて，これだけではどの方法をとるのが最もよいかははっきりしません．しかしこのゲームの場合，基地の位置だけでなく，ICBMの位置(各国4機)や各ICBMが破壊されているかいないかを示す変数(各4個)も必要になります．これだけの変数があると，②の「変数総入れ換え法」もなかなかめんどうになります．また①の方法ではプログラムが長くなるのですすめられません．そこで，③の配列を使ったやり方でプログラムを作ることにしました．

これだけでは，まだ配列を使うメリットがどういうことかよくわからない人もいるでしょう．しかし，実際のプログラムとその変数リストを見れば，十分わかっていただけると思うので，ここでは説明を控えておきます．

2．衝突をどう判定するか？

各国のICBMも基地も1つのドットではありません．ICBMは横2×縦2ドットですし，基地は横4×縦4ドットになっています．そこで，さきほどあげたように，単に両者の座標の値が等しければ衝突がおこるというものではなく，もう少し幅があるわけです．**図11**のように基地とICBMのそれぞれの代表点をとって考えると，基地とICBMが重なるのは，

図11



ΔX＝(基地のX座標)－(ICBMのX座標)

ΔY＝(基地のY座標)－(ICBMのY座標)

としたときに，

$$\begin{cases} -2 < \Delta X < 2 \\ -2 < \Delta Y < 2 \end{cases} \Rightarrow \begin{cases} ABS(\Delta X) < 2 \\ ABS(\Delta Y) < 2 \end{cases}$$

が成り立っているときです.ここで出てきたABSというのは絶対値をとる関数で，入ってきた値がマイナスならプラスになおす働きがあります．またICBMどうしが重なっているのは，同様に，

$$\begin{cases} -1 < \Delta X < 1 \\ -1 < \Delta Y < 1 \end{cases} \Rightarrow \begin{cases} ABS(\Delta X) < 1 \\ ABS(\Delta Y) < 1 \end{cases}$$

が成り立っているときになります．

以上を利用すれば，ICBM対基地，ICBM対ICBMの衝突を判定することができます．

アプローチ5 全体の流れ図を書く

図12のゼネラルフローチャートを見てください．この図をもとに，わかりにくい部分を解説します．

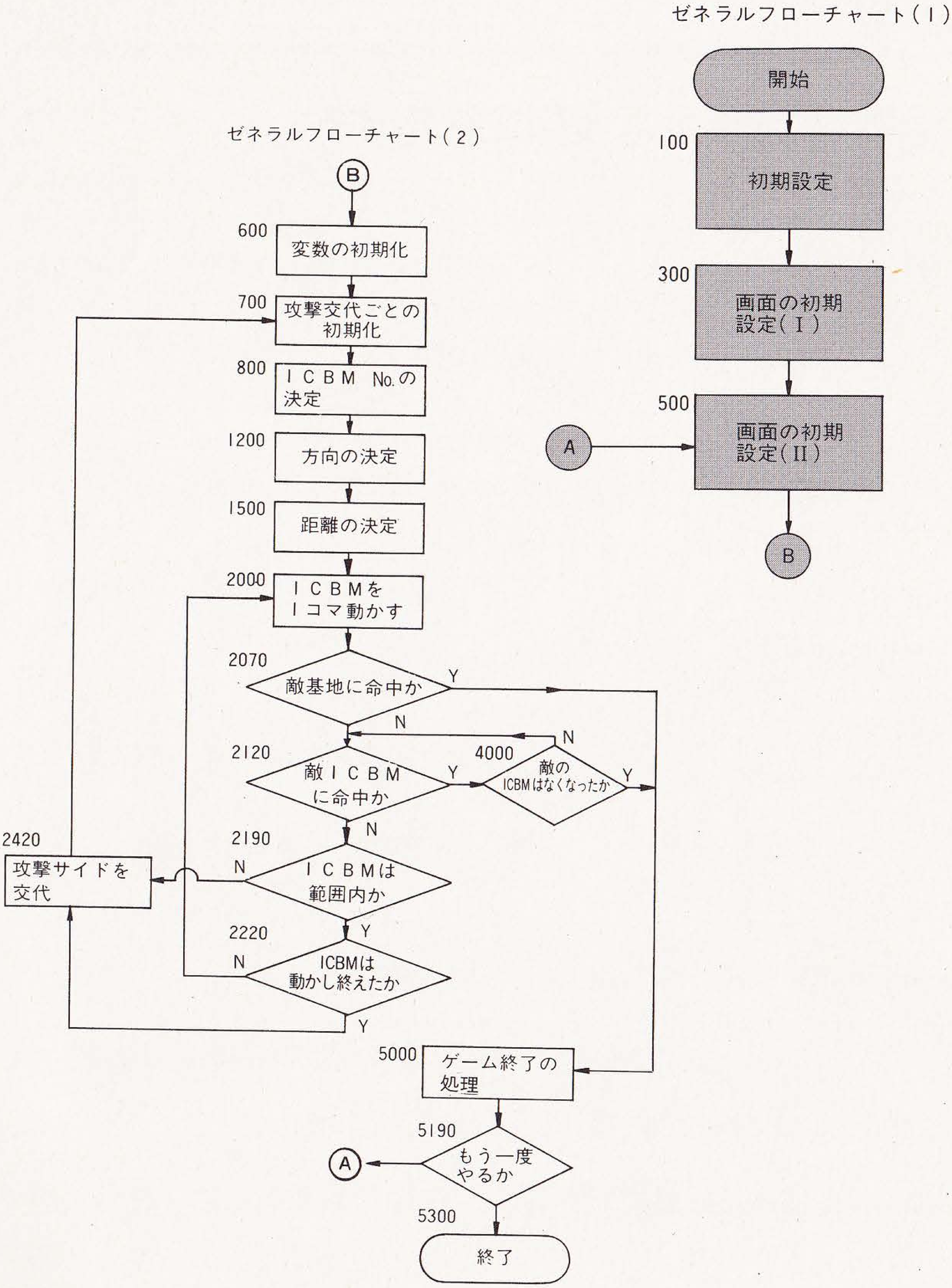
まず各枠にそえられた数字ですが，これは実際にできあがったプログラムの行番号を示しています．あとに載っている全体のプログラムを参照してみるとよいでしょう．

最初に「初期設定」，「画面の初期設定(Ⅰ)，(Ⅱ)」と並んでいますが，初期設定というのは，準備作業のことです．画面の初期設定が2つに分かれているのは，(Ⅰ)のほうは，1度やってしまえばそれ以上はやらなくてもよい部分，(Ⅱ)のほうは，ゲームを繰り返すたびに再設定しなければいけない部分を意味しています．700行の「攻撃交代ごとの初期化」のところから，2220行の「ICBMは動かし終えたか」のところまでがメインルーチンです．

なお，このフローチャートには書かれていませんが，プログラムには，一定時間ごとに割り込みがかかるタイマー割り込みを使っています．割り込みというのは，プログラムの実行中に，ある条件が整うとそれまでの作業をいったん中止して，急に別の作業に移ることをさします．そして割り込みによる別の作業が終わると，何ごともなかったかのようにまたもとのプログラムを実行していきます．

このゲームでは，5/6秒ごとに割り込みをかけ，割り込みルーチンのなかでは，時間計の表示と時間切れの判定を行ないます．ただ，この場合，通常の割り込みとちがって，時間切れのときには強制的に攻撃サイドの交代を行なうようにしています．

図12


ゼネラルフローチャート(1)
開始
100
初期設定
300
画面の初期設定(Ⅰ)
500
画面の初期設定(Ⅱ)
A
B
ゼネラルフローチャート(2)
B
600
変数の初期化
700
攻撃交代ごとの初期化
800
ICBM No.の決定
1200
方向の決定
1500
距離の決定
2000
ICBMを1コマ動かす
2070
敵基地に命中か
Y
N
2120
敵ICBMに命中か
Y
N
4000
敵のICBMはなくなったか
N
Y
2190
ICBMは範囲内か
N
Y
2220
ICBMは動かし終えたか
N
Y
2420
攻撃サイドを交代
5000
ゲーム終了の処理
5190
もう一度やるか
A
5300
終了

3節 プログラムを組む

アプローチ1 主な変数を設定する

配列を多用し，割り込み処理も行なっているこのプログラムの場合，各変数の意味をはじめにはっきりさせておくことは重要です．割り込み処理中に用いる変数と，他で使う変数をごちゃまぜにしたりすると，悪質なバグ(原因がなかなかわからないものやいくつかの条件が重なりあって突然起きるもの)が発生するもととなります．それでは，変数リストをあげてみましょう．

変数リスト

S……順番を表わす．0ならソ連の攻撃中，1ならアメリカの攻撃中．

Z……ICBMの番号(1～4の値を持つ)．

F……攻撃できる残りの時間．これが0になると，タイマー割り込みルーチンにより強制的に攻撃サイドがチェンジする．

BX(i)……i番目の国の基地の横座標．

BY(i)……i番目の国の基地の縦座標．

X(i，j)……i番目の国の(j＋1)番目のICBMのある位置の横座標．
(例)　ソ連の3番目のICBMの横座標──→X(0，2)

Y(i，j)……i番目の(j＋1)番目のICBMのある位置の縦座標．

T(i，j)……i番目の国の(j＋1)番目のICBMが破壊されていれば1，そうでなければ0を値に持つ．

S(i)……i番目の国の破壊されていないICBMの数．

N＄(i)……i番目の国の国名．
すなわちN＄(0)＝〝USSR〞，N＄(1)＝〝USA〞となる．

P……ICBMの航続距離

C……「※」型スプライトの色

L……ソ連の勝った回数

R……アメリカの勝った回数

アプローチ2 画面を作る

このゲームのように，画面にいろいろなものを描くプログラムでは，画面関係のルーチンを作ってしまうと，もうできたような気になってほっとします．ほっとするのがいいかどうかは別として，画面を描く部分のプログラムを作っている間に何かアイデアが湧いて，最初考えていたものよりもよいものを作れることもあります．手順としては，まず画面関係のルーチンを作ることをおすすめします．

リスト 9 に示したのが，初期画面を作るプログラムです．このプログラムではグラフィックス画面に文字を書くことを多用しています．そこでこの方法について解説を加えましょう．

リスト9

```
180 OPEN"GRP:" FOR OUTPUT AS #1
200 SCREEN 2,0,0
300 REM か゛めん しょきせってい ルーチン
310 COLOR 15,14,14:CLS
320 LINE(176,130)-(209,159),,BF
330 GOSUB 8000
340 LINE(22,180)-(73,189),,BF
350 LINE(24,182)-(71,187),7,BF
360 DRAW"BM92,7":PRINT#1,"N"
370 DRAW"BM14,84":PRINT#1,"W"
380 DRAW"BM170,84":PRINT#1,"E"
390 DRAW"BM92,161":PRINT#1,"S"
400 DRAW"BM204,9":PRINT#1,"N"
410 DRAW"BM204,83":PRINT#1,"S"
420 DRAW"BM178,121":PRINT#1,"ICBM"
430 DRAW"BM234,108":PRINT#1,"P"
440 DRAW"BM234,122":PRINT#1,"O"
450 DRAW"BM234,136":PRINT#1,"W"
460 DRAW"BM234,150":PRINT#1,"E"
470 DRAW"BM234,164":PRINT#1,"R"
480 DRAW"BM32,170":PRINT#1,"TIME"
500 REM スタート の し゛ゅんひ゛
510 LINE(24,16)-(167,159),,BF
520 LINE(42,126)-(57,141),3,BF
530 LINE(48,132)-(51,135),12,BF
```

```
540 LINE(134,34)-(149,49),9,BF
550 LINE(140,40)-(143,43),6,BF
560 COLOR 14
570 DRAW"BM34,144":PRINT#1,"USSR"
580 DRAW"BM130,24":PRINT#1,"USA"
999 GOTO 999
8000 REM が めんしょきか サブ ルーチン
8010 CIRCLE(208,50),32,7
8020 PAINT(208,50),7,7
8030 LINE(178,132)-(207,157),5,BF
8040 LINE(220,100)-(231,179),5,BF
8050 RETURN
```

MSXでは，グラフィックスの画面に文字を書くことを，グラフィックス画面という1つのファイル文字を出力するというふうにとらえます．こういうと難しく聞こえますが，順を追って考えればわかります．

まずファイルを使うためには，ファイルを使用するという宣言をしなければなりません．これが180行の文で，1番のファイルを"GRP:"，つまりグラフィックス画面のために使うと宣言しているのです．

次に200行では，実際に画面をグラフィックス・モードに設定しています．これではじめて，グラフィックス画面に文字が書けるようになるのです．

しかしまだこれだけでは好きな場所に文字を出すことはできません．実際に文字を書く場所は「参照点」とか「LP(ラストポイント)」とか呼ばれる点から書かれます．これは何かというと，いちばん新しく実行したグラフィックス関係の命令が関係した点です．といってもまだわかりにくいですね．具体例で見ましょう．

リスト9の320行を実行すると，(209, 159)の点まで線を引いているので，この場合はLPは(209, 159)の点になります．このようにLINE命令では最後の点，CIRCLE命令では中心の点，PSET・PRESET・POINTなどでは扱う点がLPになります．このプログラムでは，360行から480行までの間で，LPを移動させるために，

DRAW"BM▽▽,△△"

を多用しています．

さて,好きな場所に文字を出すには，このDRAW命令やその他のグラフィックス関係の命令を使って，LPを文字の書きたいところの左上にもっていきます．そして,

PRINT#1，"……"

を実行すればよいのです．

このようにグラフィックス画面に文字を描くためには，めんどうな手続きが必要なのですが，一方で，グラフィックス画面を使うことにより，多数の色を使ったり，図形と文字を同じ画面に表示できるというメリットも出てきます．ゲーム作りをこころざす人はぜひともマスターしてほしいテクニックです．

アプローチ3 メインルーチンを作る

前もって必要な処理をしておかずに，いきなりメインルーチンを組んでも，それだけではプログラムは動きません．その前に配列を定義し，必要な値を設定したり，スプライトのパターンを決めたりしておくことが必要です．このようなルーチンのことを，初期化ルーチンとかイニシャライズルーチンと呼びます．**リスト10**がそれです．220行から290行までがスプライトの設定ルーチンです．9000行からのDATA文から１つずつ値を読み込んできて，８文字をひとまとめにして１つのスプライト・パターンを作っています．ちょっと見るとわかりにくそうですが，49ページの図10と見比べながら考えればわかると思います．

リスト10

```
100 REM しょきせってい ルーチン
120 DEFINT C,E-W,Z:DEFSNG A,D,X,Y
130 DIM N(1),BX(1),BY(1),S(1)
140 DIM X(1,3),Y(1,3),T(1,3)
150 BX(0)=49:BY(0)=133
160 BX(1)=141:BY(1)=41
170 N$(0)="USP":N$(1)=" USA"
180 OPEN"GRP:" FOR OUTPUT AS #1
```

```
200 SCREEN 2,0,0
210 REM ｽﾌﾟﾗｲﾄ ﾉ ｾｯﾃｲ
220 FOR I=0 TO 4
230 A$=""
240 FOR J=1 TO 8
250 READ B$
260 A$=A$+CHR$(VAL("&H"+B$))
270 NEXT J
280 SPRITE$(I)=A$
290 NEXT I
600 FOR I=0 TO 3:X(0,I)=49:Y(0,I)=133:NE
XT
610 FOR I=0 TO 3:X(1,I)=141:Y(1,I)=41:NE
XT
620 FOR I=0 TO 3:T(0,I)=0:T(1,I)=0:NEXT
630 S(0)=4:S(1)=4
640 A=RND(-TIME)
650 IF A>.5 THEN S=1 ELSE S=0
9000 REM ｽﾌﾟﾗｲﾄ ﾉ ﾃﾞｰﾀ
9010 DATA 00,92,44,28,92,28,44,92
9020 DATA C0,C0,00,02,06,02,02,02
9030 DATA C0,C0,00,03,05,01,02,07
9040 DATA C0,C0,00,07,01,07,01,07
9050 DATA C0,C0,00,08,0A,0A,0F,02
```

その他の部分は全リストに載せた説明を見ればおよそ見当がつくでしょう.

さてそれではいよいよメインルーチンにはいりましょう. さきほどのバリケードゲームの場合, メインルーチンは小さかったので, 一度にリストに出してもよかったのですが, 今度はたいへんです. まずは, 図12のゼネラルフローチャートを振り返ってみてください.ここで枠で囲ってあるものごとに見ていきましょう. まずは攻撃交代ごとの初期化というルーチンですが,**リスト11**を見てください.ここでは時間計を初期状態(時間が十分残っている状態)にする作業と,攻撃例がどちらかを表示する作業の2つを行なっています.

リスト11

```
700 REM ﾒｲﾝ ﾙｰﾁﾝ
710 LINE(24,182)-(71,187),10,BF
```

```
720 F=47
730 REM し゛ゅんは゛んの ひょうし゛
740 LINE(199,185)-(112,176),14,BF
750 PRINT#1,N$(S);"の は゛ん"
```

1. ICBM番号決定ルーチン

次にICBMの番号を決定するルーチンですが,**リスト12**がそれです.変数Cは「※」印スプライトを表示する色をあらわす変数です.「※」印スプライトを表示するのは,プレイヤーに入力を促すためです.840行を実行すると,Cの値は0と15とが交互に入れ換わり,したがって,スプライトは点滅して表示される(透明と白が入れ換わる)わけです.また850～890行でFOR～NEXTを使ってループさせている(繰り返させている)のは,「※」印スプライトの点滅の速さを調整するためです.900行は,ICBMの番号を示す変数Zに値が代入されていなければ,RETURNを押しただけでは次に進めないようにするものです.このプログラムを読めばわかるよ

リスト12

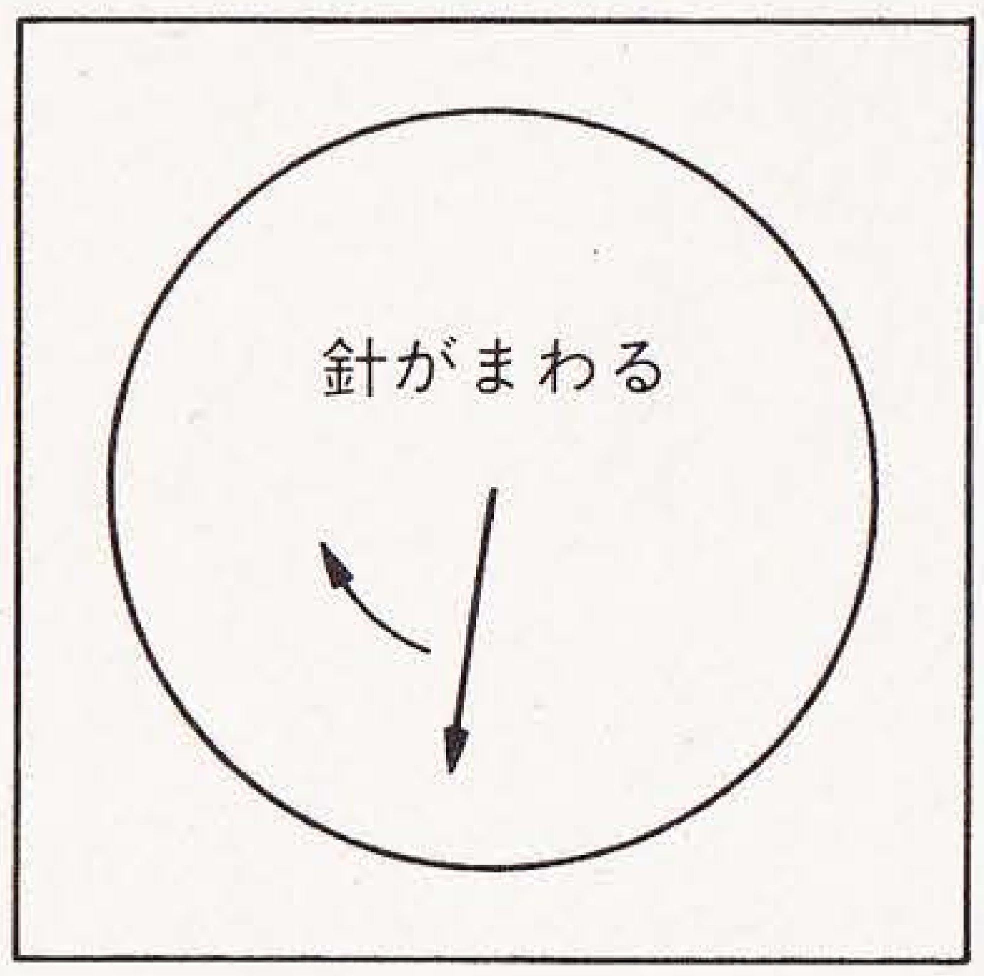
```
800 REM た゛んか゛んの は゛んこ゛うを きめる
820 C=15:Z=0
830 PUT SPRITE0,(200,132),C,0
512 C=15-C
850 FOR I=1 TO 30
860 A$=INKEY$
870 IF A$<>"" THEN 900
880 NEXT
890 GOTO 1000
900 IF A$=CHR$(13) AND Z THEN 1200
910 J=VAL(A$)
920 IF J<1 OR J>4 THEN 1000
930 IF T(S,J-1) THEN BEEP:GOTO 1000
940 REM ナンハ゛ー の ひょうし゛
960 LINE(197,149)-(190,142),5,BF
970 PRINT#1,USING"#";J
990 Z=J
1000 GOTO 830
1200 REM ほうこうを きめる ルーチン
```

うに，ICBMの番号の入力法は，まず番号（1～4）を入力し，確認のために，さらに RETURN を押すようにしてあります．

2．方向決定ルーチン

さて次はICBMの飛行方向を決めるルーチンです．**リスト13**を見てください．このルーチンは方向計の中の針をぐるぐる回転させ，RETURN を押したときの方向を変数Dに求めています．ここでもやはり1290～1320行での時間かせぎや，「※」印スプライトの点滅を行なっています．





リスト13

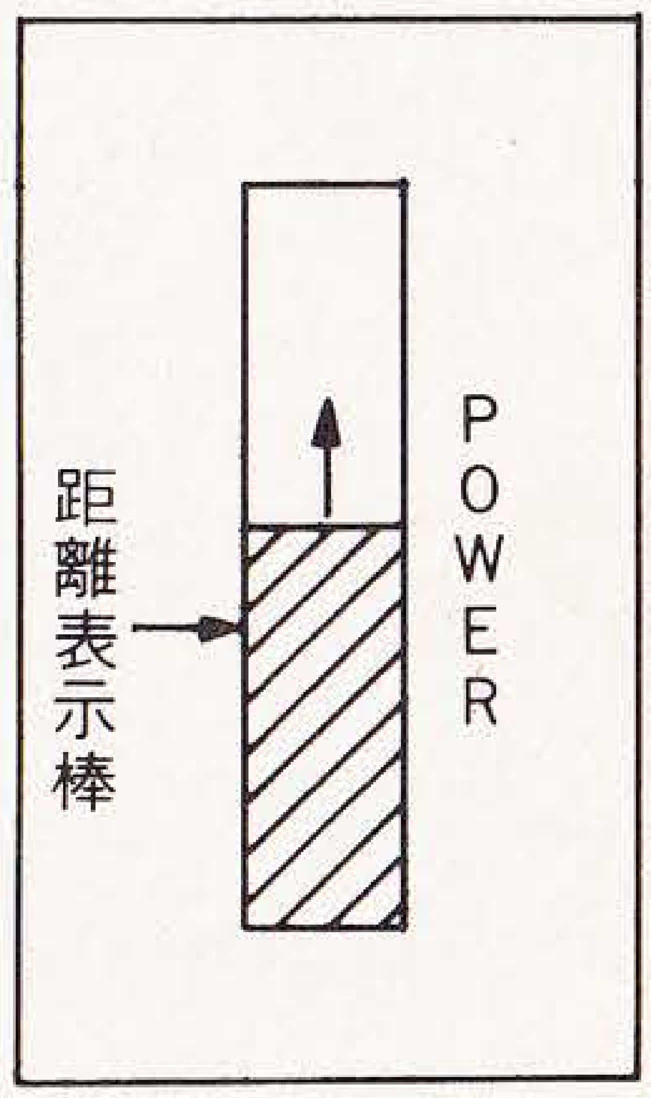
```
1200 REM ほうこうを きめる ルーチン
1210 D=RND(1)*6.2832
1220 X=208+28*COS(D2):Y=50+28*SIN(D2
1230 LINE(208,50)-(X,Y),7
1240 X=208+28*COS(D):Y=50+28*SIN(D)
1250 LINE(208,50)-(X,Y),14
1260 D2=D
1270 PUT SPRITE0,(218,33),C,0
1280 C=15-C
1290 FOR I=1 TO 30
1300 A$=INKEY$
1310 IF A$<>"" THEN 1340
1320 NEXT
1330 GOTO 1350
1340 IF A$=CHR$(13) THEN 1500
1350 D=D+.25
1360 GOTO 1230
1500 REM きょりを きめる ルーチン
```

3．距離決定ルーチン

今度は**リスト14**をご覧ください．これはICBMの飛行距離を決めるルーチンです．このルーチンが実行されると，距離計の中を距離表示棒がすばやく上下します．そしてプレイヤーが[RETURN]を押した時点で距離表示棒が止まり，その長さが飛行距離となります．距離表示棒を上下させるルーチンは1530～1560行です．1540行で，[RETURN]が押されたときに1590行に飛ぶようにしています．1590行には，INTERVAL OFFという１行があります．これはタイマー割り込みを中止する命令ですが，今までタイマー割り込みについての説明はしないできました．メインルーチンはまだ続きますが，ここでちょっとタイマー割り込みルーチンについてお話しましょう．

リスト14

```
1500 REM きょりを きめる ルーチン
1510 PUT SPRITE0,(0,0),0,0
1520 LINE(220,100)-(231,179),5,BF
1530 FOR I=2 TO 80 STEP 2
1540 LINE(220,180-I)-(231,181-I),9,BF
1550 A$=INKEY$
1560 IF A$=CHR$(13) THEN 1590
1570 NEXT
1580 GOTO 1520
1590 INTERVAL OFF
1600 P=I/2
```



4．タイマー割り込みルーチン

リスト15がタイマー割り込みルーチンです．変数Fは残り時間を示すもので，タイマー割り込みがかかるたびに7010行で１つずつ減らしていきます．そして時間計の水色部分(消費時間を示す)を増やし，Fが0より大きければそれで戻ります．そしてFが0になると，強制的に順番を交代するために2420行へ戻ります．

リスト15

```
7000 REM タイマー わりこみ ルーチン
7010 F=F-1
```

```
7020 IF F<=0 THEN F=0:GOTO 7050
7030 LINE(24,182)-(71-F,187),7,BF
7040 RETURN
7050 LINE(199,185)-(112,178),14,BF
7060 PRINT#1,"TIME OVER !"
7070 FOR I=1 TO 20:BEEP:NEXT
7080 F=47
7090 RETURN 2420
```

このタイマー割り込みの発生・停止を決めるのは，**リスト16**の３つの行です．190行は5/6秒ごとに7000行にタイマー割り込みがかかると定義したもので，これは実際には810行のINTERVAL ONを実行してはじめて有効になります．1590行で割り込みを止めているのは，810～1580行の間のルーチンでICBMの番号，方向，距離を入力し終わったからです．

リスト16

```
190 ON INTERVAL=50 GOSUB 7000
810 INTERVAL ON
1590 INTERVAL OFF
```

また，**リスト17**を見てください．950行で割り込みに停止をかけている理由は，960行と970行の間に割り込みがかかると，割り込みルーチン中でグラフィックスの参照点(LP)が移動してしまい970行で正しい場所に字が書けなくなるからです．

リスト17

```
950 INTERVAL STOP
960 LINE(197,149)-(190,142),5,BF
970 PRINT#1,USING"#";J
980 INTERVAL ON
```

5．ICBM移動ルーチン

さて，メインルーチンのほうに話を戻しましょう．さきほどまでの部分で，

ICBMを動かすのに必要な番号・方向・距離という3つの要素を求めた(各々変数Z, D, Pにはいっている)わけですが，いよいよ今度はそのICBMを動かします.

リスト18をご覧ください. これがICBMを動かす部分のルーチンです. 実際にICBMを動かすのは2030～2060行の部分だけなのですが，ゼネラルフローチャートを見ればわかるように，いろいろと判定すべきことが多いので複雑になっています. 2060行でスプライト画面の番号がS＊4＋Zとなっていますが，SとZの値により，両国の計8機のICBMが1～8番までに区別できるわけです. また，配列変数の意味は取りちがえないように注意してください. 添字の部分で，(S)とあればそれは攻撃側のもので，(1－S)とあれば防御側のものと考えてよいでしょう. その他はゼネラルフローチャートを見ながら考えればおよそ理解できると思います.

リスト18

```
2000 REM だ゛んか゛ん を うご゛かす ルーチン
2010 FOR I=1 TO P
2020 PLAY"T240o3G16"
2030 X(S,Z-1)=X(S,Z-1)+COS(D)
2040 Y(S,Z-1)=Y(S,Z-1)+SIN(D)
2050 X=X(S,Z-1):Y=Y(S,Z-1)
2060 PUT SPRITE S*4+Z,(X,Y),7,Z
2070 REM てきの きち との しょうとつ はんてい
2080 A=X-BX(1-S)
2090 IF ABS(A)>2 THEN 2120
2100 A=Y-BY(1-S)
2110 IF ABS(A)<=2 THEN 5000
2120 REM てきの だ゛んか゛ん との しょうとつ はんてい
2130 FOR J=0 TO 3
2140 A=X-X(1-S,J)
2150 IF ABS(A)>1 THEN 2180
2160 A=Y-Y(1-S,J)
2170 IF ABS(A)<=1 THEN 4000
2180 NEXT J
2190 REM はんいか゛いに て゛たか はんてい
2200 IF X<24 OR X>166 THEN 3000
2210 IF Y<16 OR Y>158 THEN 3000
2220 NEXT I
2400 PUT SPRITE 4*S+Z,(X,Y),2+6*S,Z
2410 PLAY"T240L8O3CCGFEDCC"
```

```
2420 S=1-S
2430 GOSUB 8000
2440 IF PLAY(0) THEN 2440
2450 GOTO 700
3000 REM はんいかﾞいに てﾞた
4000 REM てき たﾞんかﾞんに めいちゅう
5000 REM てき きちに めいちゅう
8000 REM かﾞめんしょきか サブﾞルーチン
8010 CIRCLE(208,50),32,7
8020 PAINT(208,50),7,7
8030 LINE(178,132)-(207,157),5,BF
8040 LINE(220,100)-(231,179),5,BF
8050 RETURN
```

さて,メインルーチンが終わり,あとにICBMを破壊したときの処理, ICBMが範囲外に出たときの処理, 敵基地を破壊したときの処理が残りました. これらは全リストに一括して示すことにします. **リスト19**中の3000～5300行がこれらの処理をする部分です.

これで3章の終わりとしますが, 皆さんには**リスト19**の全プログラムを見て, まず自分の機械でプログラムを走らせてみることをおすすめします. 実際にゲームをしてから考えれば, これまでに述べてきたことの意味もよりはっきりすることでしょう. そして, プログラム作りのステップ (アイデアにはじまり, 構想, 流れ図, プログラムとなる) を思い起こしつつ, プログラムの内容を把握できたら, もうゲーム作りのノウハウの大事な部分は確実に修得できていると言えるでしょう.

リスト19　南極米・ソ戦ゲーム全リスト

```
10 REM
20 REM  Antarctic War  USA vs USSR
30 REM
40 REM         Ver 1.00
50 REM
100 REM しょきせってい ルーチン
120 DEFINT C,E-W,Z:DEFSNG A,D,X,Y
```

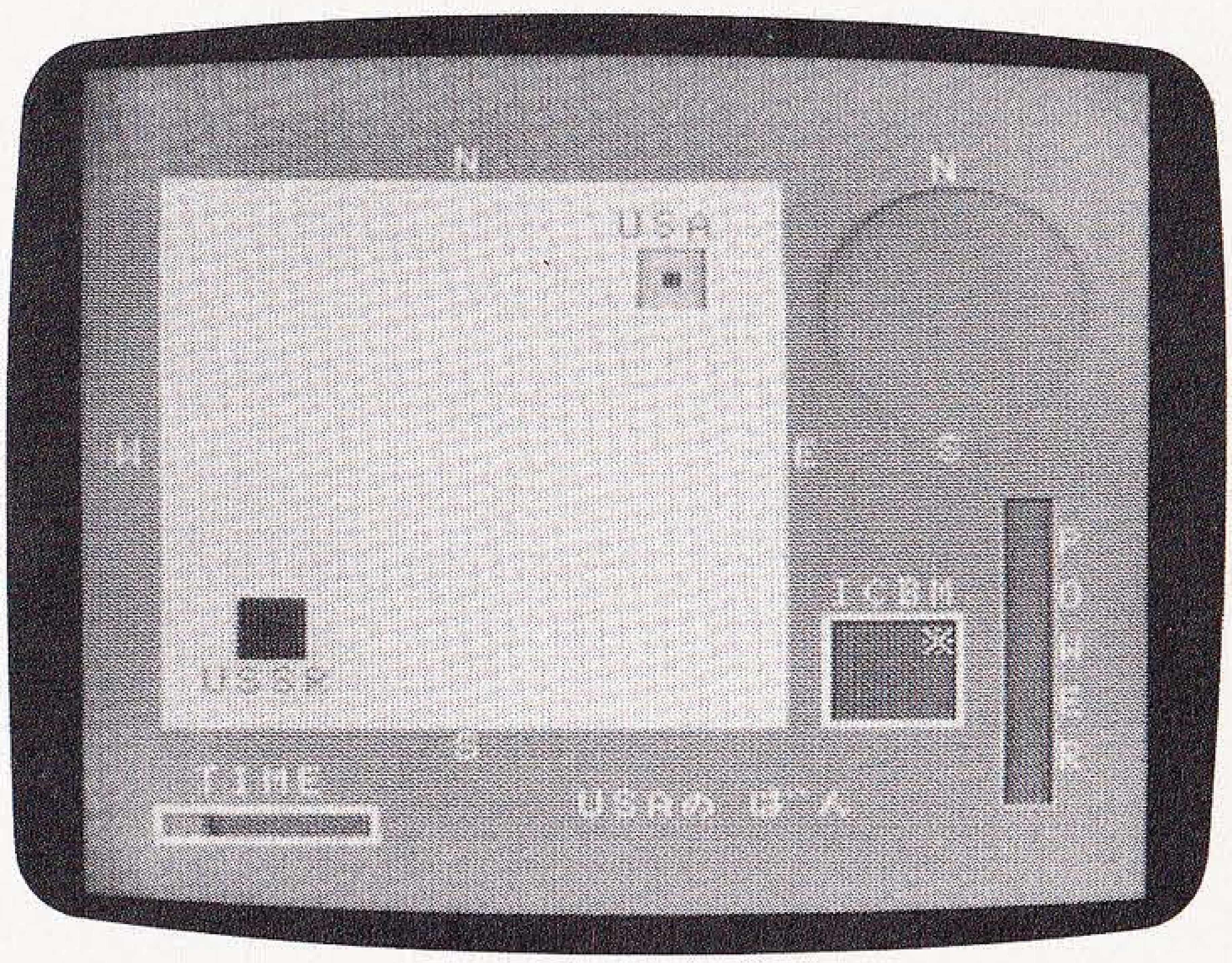
```
130 DIM N(1),BX(1),BY(1),S(1)    } 配列変数の定義
140 DIM X(1,3),Y(1,3),T(1,3)     }
150 BX(0)=49:BY(0)=133           } 基地の位置の設定
160 BX(1)=141:BY(1)=41           }
170 N$(0)="USSR":N$(1)=" USA" ←国名の設定
180 OPEN"GRP:" FOR OUTPUT AS #1
190 ON INTERVAL=50 GOSUB 7000 ←タイマー割り込みの設定
200 SCREEN 2,0,0
210 REM ｽﾌﾟﾗｲﾄ の せってい
220 FOR I=0 TO 4
230 A$="" ←変数の内容をまずはなくす
240 FOR J=1 TO 8
250 READ B$ ←データから1つ取ってくる
260 A$=A$+CHR$(VAL("&H"+B$))
270 NEXT J
280 SPRITE$(I)=A$
290 NEXT I
300 REM がめん しょきせってい ﾙｰﾁﾝ
310 COLOR 15,14,14:CLS
320 LINE(176,130)-(209,159),,BF ←ICBM表示窓の枠を描く
330 GOSUB 8000
340 LINE(22,180)-(73,189),,BF    } 時間計を描く
350 LINE(24,182)-(71,187),7,BF   }
360 DRAW"BM92,7":PRINT#1,"N"
370 DRAW"BM14,84":PRINT#1,"W"    } 戦場の周囲の方位を書く
380 DRAW"BM170,84":PRINT#1,"E"   }
390 DRAW"BM92,161":PRINT#1,"S"   }
400 DRAW"BM204,9":PRINT#1,"N"    } 方向計の上下の方位を書く
410 DRAW"BM204,83":PRINT#1,"S"   }
420 DRAW"BM178,121":PRINT#1,"ICBM"
430 DRAW"BM234,108":PRINT#1,"P"
440 DRAW"BM234,122":PRINT#1,"O"
450 DRAW"BM234,136":PRINT#1,"W"
460 DRAW"BM234,150":PRINT#1,"E"
470 DRAW"BM234,164":PRINT#1,"R"
480 DRAW"BM32,170":PRINT#1,"TIME"
500 REM ｽﾀｰﾄ の じゅんび
510 LINE(24,16)-(167,159),,BF          }
520 LINE(42,126)-(57,141),3,BF         }
530 LINE(48,132)-(51,135),12,BF        } 戦場内の初期画面を作る
540 LINE(134,34)-(149,49),9,BF         }
550 LINE(140,40)-(143,43),6,BF         }
560 COLOR 14                           }
570 DRAW"BM34,144":PRINT#1,"USSR"      }
```

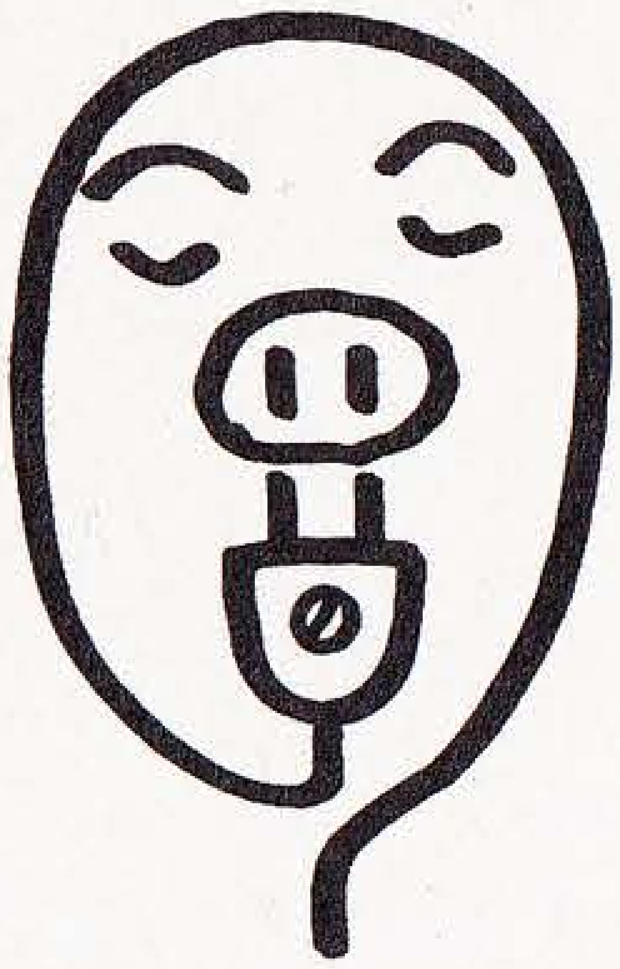
```
580 DRAW"BM130,24":PRINT#1,"USA"
590 COLOR 15
600 FOR I=0 TO 3:X(0,I)=49:Y(0,I)=133:NEXT  } ICBMの位置の
610 FOR I=0 TO 3:X(1,I)=141:Y(1,I)=41:NEXT  } 初期化
620 FOR I=0 TO 3:T(0,I)=0:T(1,I)=0:NEXT     }
630 S(0)=4:S(1)=4  ←ICBMの数の初期化
640 A=RND(-TIME)                    } 先攻の国を乱数で決定
650 IF A>.5 THEN S=1 ELSE S=0       }
700 REM メイン ルーチン
710 LINE(24,182)-(71,187),10,BF  } 時間計をいっぱいにしておく
720 F=47                         }
730 REM じゅんばんの ひょうじ
740 LINE(199,185)-(112,178),14,BF
750 PRINT#1,N$(S);"の ばん"
800 REM だんがんの ばんごうを きめる
810 INTERVAL ON
820 C=15:Z=0
830 PUT SPRITE0,(200,132),C,0  ←※印スプライトの表示
840 C=15-C  ←※印の色を透明←→白に変える
850 FOR I=1 TO 30
860 A$=INKEY$              } ウェイトを入れるために使ったFOR～NEXT
870 IF A$<>"" THEN 900     }
880 NEXT
890 GOTO 1000
900 IF A$=CHR$(13) AND Z THEN 1200
910 J=VAL(A$)
920 IF J<1 OR J>4 THEN 1000
930 IF T(S,J-1) THEN BEEP:GOTO 1000
940 REM ナンバー の ひょうじ
950 INTERVAL STOP
960 LINE(197,149)-(190,142),5,BF  } ICBMナンバーの表示
970 PRINT#1,USING"#";J            }
980 INTERVAL ON
990 Z=J  ←変数ZをICBMナンバーとする
1000 GOTO 830
1200 REM ほうこうを きめる ルーチン
1210 D=RND(1)*6.2832  ←最初の方向を乱数で決める
1220 X=208+28*COS(D2):Y=50+28*SIN(D2)  } 前に描いた線を消す
1230 LINE(208,50)-(X,Y),7              }
1240 X=208+28*COS(D):Y=50+28*SIN(D)  } 新しい線を描く
1250 LINE(208,50)-(X,Y),14           }
1260 D2=D
1270 PUT SPRITE0,(218,33),C,0  ←※印スプライトの表示
1280 C=15-C  ←※印スプライトの色を変える
```

```
1290 FOR I=1 TO 30
1300 A$=INKEY$
1310 IF A$<>"" THEN 1340
1320 NEXT
1330 GOTO 1350
1340 IF A$=CHR$(13) THEN 1500
1350 D=D+.25   ←角度を加える
1360 GOTO 1230
1500 REM きょりを きめる ルーチン
1510 PUT SPRITE0,(0,0),0,0   ←※印スプライトを消去する
1520 LINE(220,100)-(231,179),5,BF   ←距離表示棒の長さを0にする
1530 FOR I=2 TO 80 STEP 2
1540 LINE(220,180-I)-(231,181-I),9,BF   ←距離表示棒を長くする
1550 A$=INKEY$
1560 IF A$=CHR$(13) THEN 1590
1570 NEXT
1580 GOTO 1520
1590 INTERVAL OFF   ←時間制限はなくす
1600 P=I/2
2000 REM だ゛んか゛ん を うご゛かす ルーチン
2010 FOR I=1 TO P
2020 PLAY"T240O3G16"
2030 X(S,Z-1)=X(S,Z-1)+COS(D) ┐
2040 Y(S,Z-1)=Y(S,Z-1)+SIN(D) ┘ ICBMの位置を進める
2050 X=X(S,Z-1):Y=Y(S,Z-1)   ←何回も使うので，短い変数に代入
2060 PUTSPRITE S*4+Z,(X,Y),7,Z
2070 REM てきの きち との しょうとつ はんてい
2080 A=X-BX(1-S)
2090 IF ABS(A)>2 THEN 2120   ┐
2100 A=Y-BY(1-S)             │ もしも衝突なら5000行へ
2110 IF ABS(A)<=2 THEN 5000  ┘
2120 REM てきの だ゛んか゛ん との しょうとつ はんてい
2130 FOR J=0 TO 3
2140 A=X-X(1-S,J)
2150 IF ABS(A)>1 THEN 2180   ┐
2160 A=Y-Y(1-S,J)            │ もしも衝突なら4000行へ
2170 IF ABS(A)<=1 THEN 4000  ┘
2180 NEXT J
2190 REM はんいか゛いに て゛たか はんてい
2200 IF X<24 OR X>166 THEN 3000 ┐
2210 IF Y<16 OR Y>158 THEN 3000 ┘ 範囲外なら3000行へ
2220 NEXT I
2400 PUT SPRITE 4*S+Z.(X,Y),2+6*S,Z
2410 PLAY"T240L8O3CCGFEDCC"
```

```
2420 S=1-S  ←攻撃サイドの交代(0←→1)
2430 GOSUB 8000
2440 IF PLAY(0) THEN 2440 ←音が出ている間は次へ進ませない
2450 GOTO 700
3000 REM はんいか゛いに て゛た
3010 PUT SPRITE S*4+Z,(-8,-8) ←スプライトを消す
3020 X(S,Z-1)=BX(S):Y(S,Z-1)=BY(S) ←ICBMを基地に戻す
3030 FOR I=1 TO 16:BEEP:NEXT
3040 GOTO 2420
4000 REM てき た゛んか゛んに めいちゅう
4010 PUT SPRITE J-S*4+5,(-8,-8) ←スプライトを消す
4020 T(1-S,J)=1                  } ICBMは破壊され，位置は(0, 0)とする
4030 X(1-S,J)=0:Y(1-S,J)=0       }
4040 S(1-S)=S(1-S)-1 ←防御側ICBM数を減らす
4050 IF S(1-S)=0 THEN 5000 ←ICBMがなくなったら5000行へ
4060 PLAY"T240L16BGAFGEFDEDCC" } ←音楽をならす
4070 IF PLAY(0) THEN 4070      }
4080 GOTO 2180
5000 REM てき きちに めいちゅう
5010 PLAY"T120L4GOCEG.E8GGO3"
5020 X=BX(1-S)-21:Y=BY(1-S)-21
5030 FOR I=1 TO 8              }
5040 PUT SPRITE I,(-8,-8)      } ICBMのスプライトの消去
5050 NEXT                      }
5060 LINE(X,Y)-(X+39,Y+39),2+6*S,BF }
5070 DRAW"BM-30,-23"                } 敗戦側にOUTと表示
5080 PRINT#1,"OUT"                  }
5090 FOR I=1 TO 4000:NEXT
5100 LINE(24,16)-(167,159),15,BF
5110 COLOR 14
5120 IF S THEN R=R+1 ELSE L=L+1 ←通算勝敗数に今の勝敗を加算
5130 DRAW"BM64,48":PRINT#1,"USSR WON"
5140 DRAW"BM64,62"                     }
5150 PRINT#1,USING"### TIMES";L;       }
5160 DRAW"BM64,80":PRINT#1,"USA  WON"  } 勝敗数の表示
5170 DRAW"BM64,94"                     }
5180 PRINT#1,USING"### TIMES";R;
5190 DRAW"BM52,120":PRINT#1,"PLAY AGAIN?"
5200 DRAW"BM76,134":PRINT#1,"(Y/N)"
5210 COLOR 15
5220 A$=INKEY$ ←大文字でも小文字でもよいように，多少の親切
5230 IF A$="N" OR A$="n" THEN5300
5240 IF A$="Y" OR A$="y" THEN5260
5250 GOTO 5220
```

```
5260 GOSUB 8000
5270 GOTO 500
5300 END
7000 REM ﾀｲﾏｰ わりこみ ﾙｰﾁﾝ
7010 F=F-1
7020 IF F<=0 THEN F=0:GOTO 7050
7030 LINE(24,182)-(71-F,187),7,BF ←時間計の水色部分を増やす
7040 RETURN
7050 LINE(199,185)-(112,178),14,BF
7060 PRINT#1,"TIME OVER !"
7070 FOR I=1 TO 20:BEEP:NEXT
7080 F=47
7090 RETURN 2420 ←強制的に攻撃サイドを交代する
8000 REM が゛めんしょきか サフ゛ﾙｰﾁﾝ
8010 CIRCLE(208,50),32,7           ┐
8020 PAINT(208,50),7,7             │ メーター類の表示の初期化
8030 LINE(178,132)-(207,157),5,BF  ┘
8040 LINE(220,100)-(231,179),5,BF
8050 RETURN
9000 REM ｽﾌﾟﾗｲﾄ の デ゛ｰﾀ
9010 DATA 00,92,44,28,92,28,44,92
9020 DATA C0,C0,00,02,06,02,02,02  ┐
9030 DATA C0,C0,00,03,05,01,02,07  │ 16進表記
9040 DATA C0,C0,00,07,01,07,01,07  │
9050 DATA C0,C0,00,08,0A,0A,0F,02  ┘
```

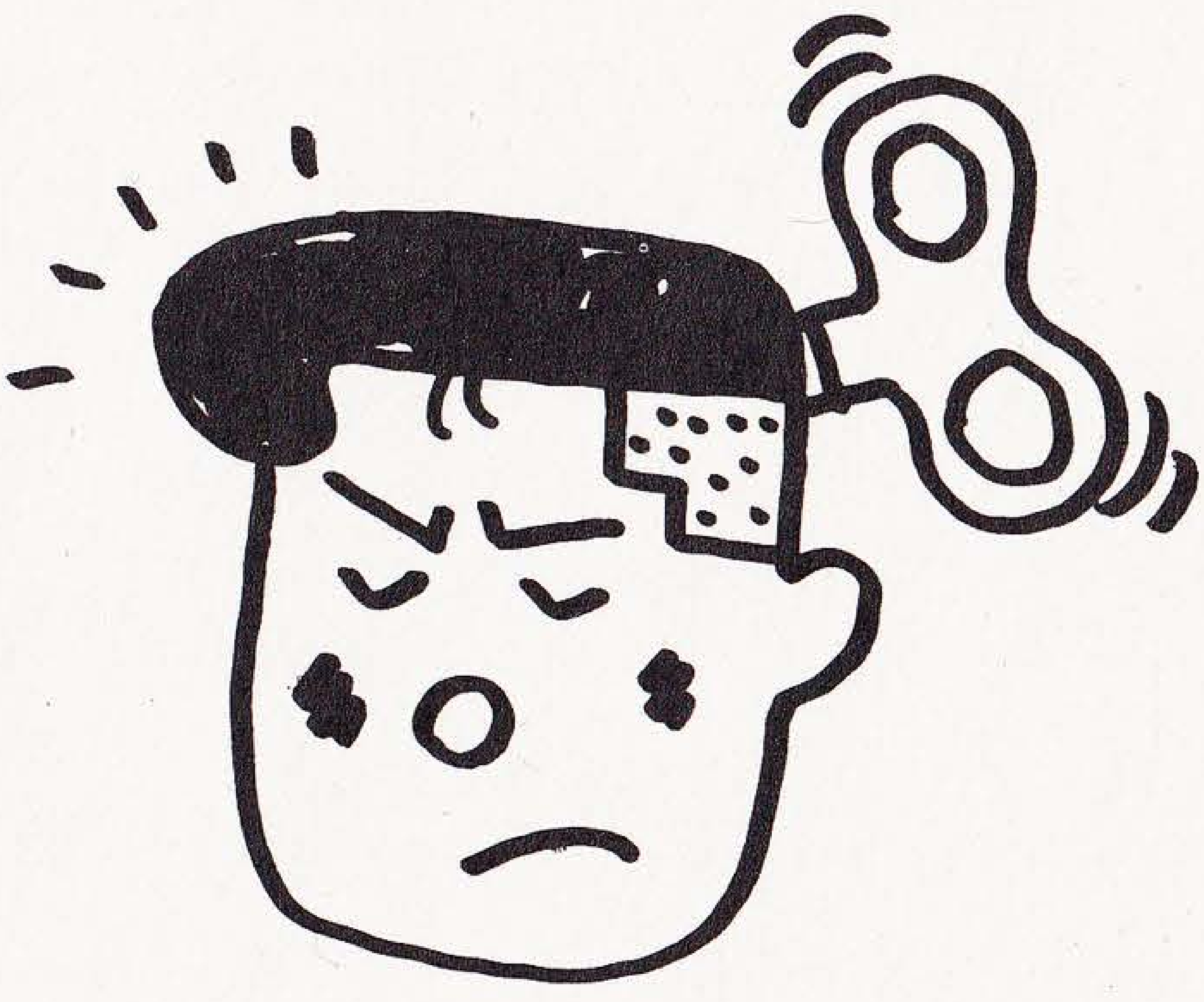




4

（実践編III）

ブロックくずし



1節 ゲームの紹介

1章1節の「ゲームあれこれ」でも紹介したように，このブロックくずしは，アーケードゲームの草分け的な存在です．現在のアーケードゲームはそれに比べると，より画面のデザインや動きにこったものが多くなりましたが，シンプルながらも熟達にはそれなりの努力とセンスが必要なこのブロックくずしのオモシロサは，時がたっても変わるものではありません．

それではブロックくずしで遊んだことのない方もいるでしょうから，簡単なルールの説明をしましょう．**図13**を見てください．三方が壁で囲まれ，上方にはたくさんのブロック，下方には1個のラケットがあります．この中を1つのボールが動きまわり，壁やラケット，ブロックに当たると反射します．そしてボールがブロックに当たったときには，そのブロックが消えます．ゲームの目的はなるべく多くの得点をかせぐ（そのためにはなるべく多くのブロックを消す）ことです．プレイヤーはラケットを左右に操作し，ボールを受け損わないようにがんばるわけです．

普通ボールはラケットとブロックの間を往復するので，ラケットで1回受けるたびにブロックも1つ消える勘定になります．しかし**図14**のように，ブロックの列のどこかに穴があいてボールが裏側のほうまではいり込むと，たくさんのブロックを1度に消すことができます．このゲームの醍醐味はこうして穴を開け，そこにボールが飛びこむようにすることにあると言えましょう．

なお，ラケットの大きさは段階を踏んでしだいに小さくなります．そして最後には点のようになるので，難しさも増していくわけです．またアーケードゲームではボールの速度もだんだん速くなるのですが，このブロックくずしでは速度は変えないようにしました．もともとブロックくずしは高速で動くマシン語で組まれているのですが，BASICだけで組んだのでは，速さの点で限界があるからです（機械語とBASICの実行速度は，ものにもよりますが，数十倍機械語のほうが速いと考えてよいでしょう）．

図13

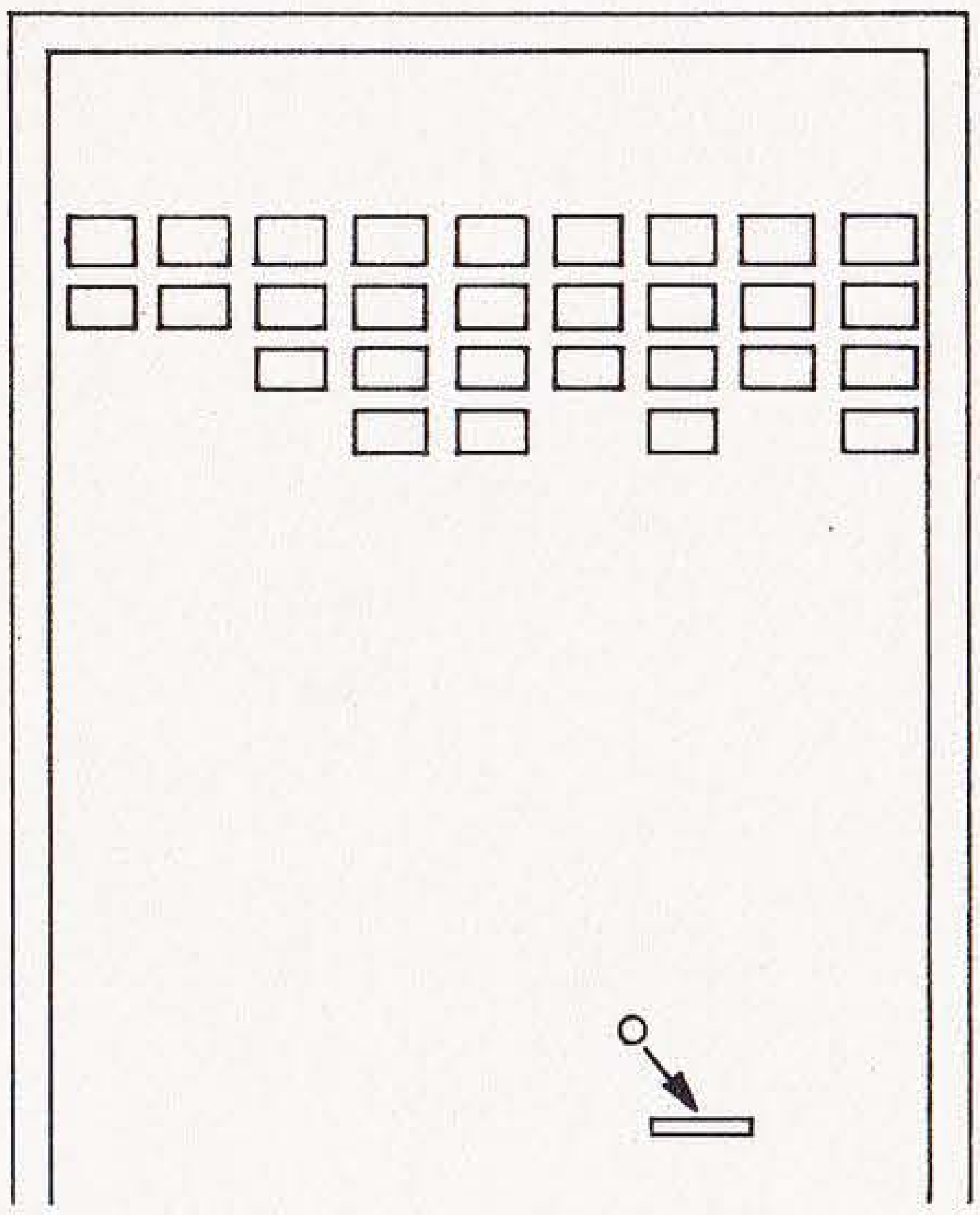
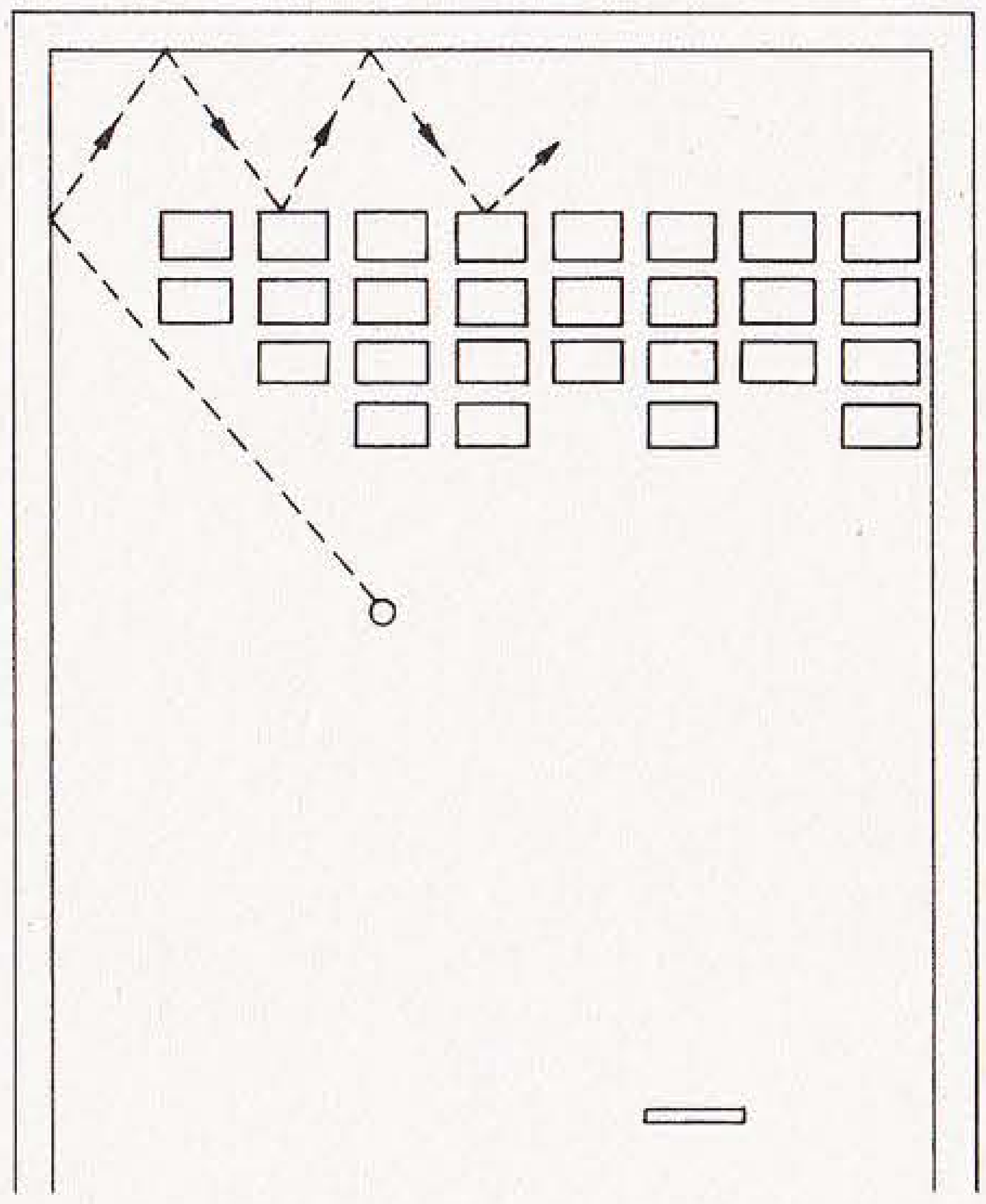


図14



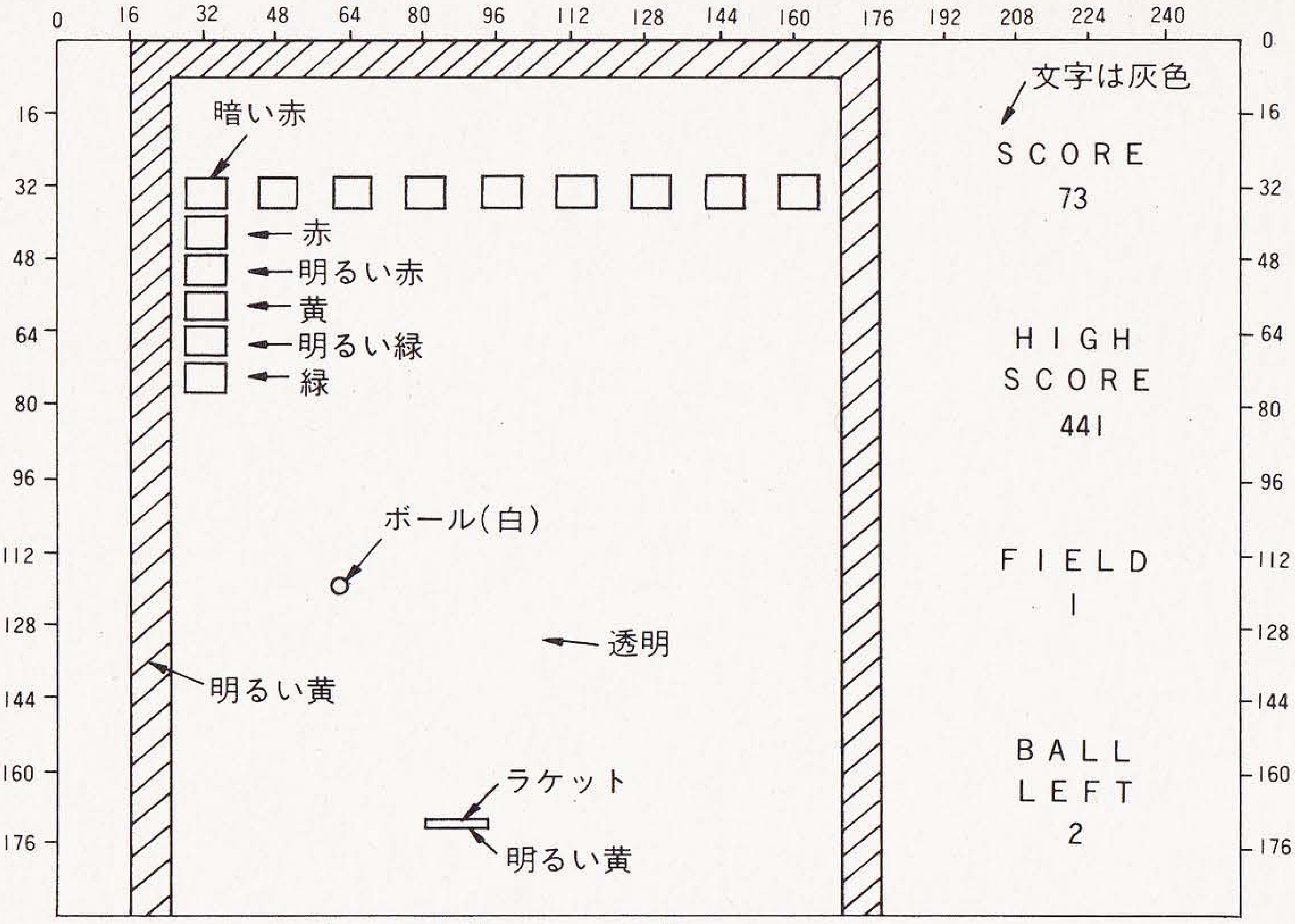
2節 ゲームの骨組みを決める

アプローチ1 レイアウトを決める

最初にスクリーンモードを決めましょう．このゲームの場合も，ボールは小さいし，カラーを豊富に使いたいので，**モード2**がよいでしょう(VDP関係の命令を使えば，モード1でもうまくできるのですが)．スクリーンレイアウトを見てください．色は黒を背景色として，地の部分は透明(したがって黒に見える)，壁は明るい黄で描きます．ブロックは横9個×縦6個並んでおり，各ブロックは横14×縦7ドットで構成されます．ブロック間には，横方向は2ドット，縦方向は1ドットのすき間をあけます．そしてブロックの色は，横1列ごとに同じ色とし，上の段から暗い赤，赤，明るい赤，黄，明るい緑，緑の6色とします．またラケットの色は壁と同じ明るい黄，ボールの色は白とします．

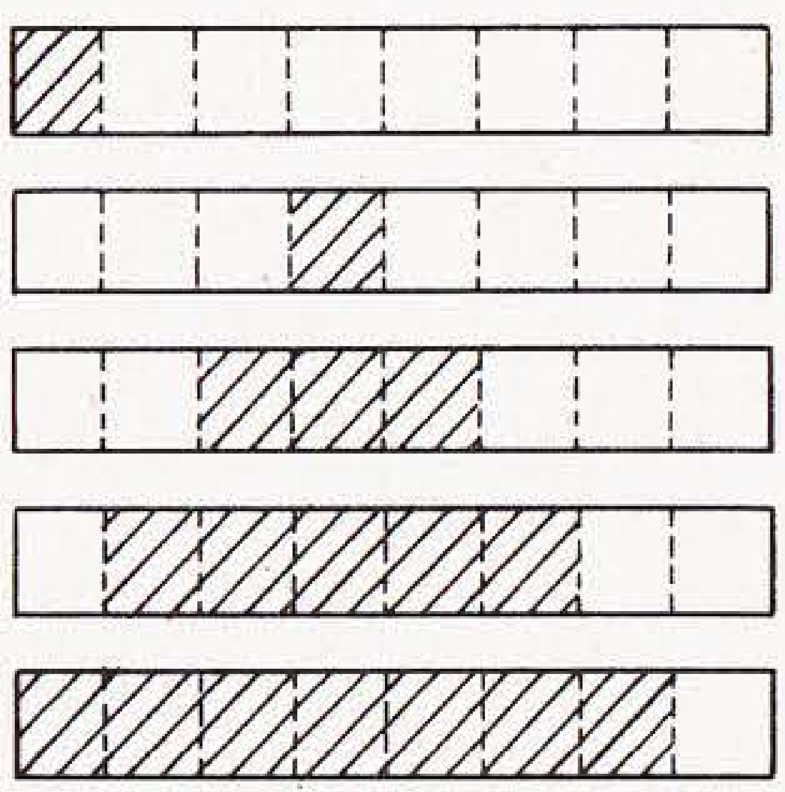
また，ボールとラケットは速く動かすものですから，やはりスプライトを使ったほうがよいでしょう．ラケットの大きさは何段階かに変わるようにするので，その数の分だけスプライト・パターンも用意しなければなりません．**図15**にあげ

ブロックくずしスクリーンレイアウト

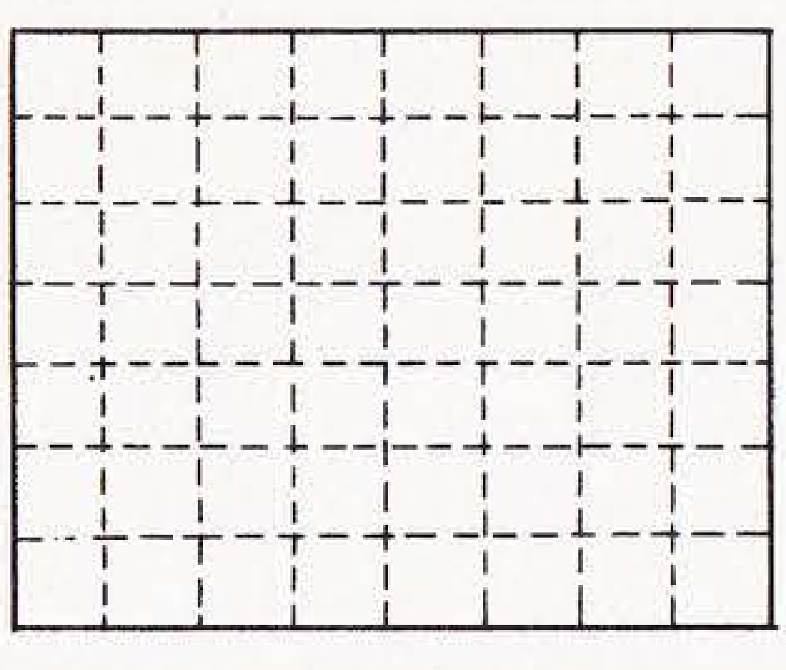


ブロックは横14ドット×縦7ドット
ブロック間のすきまは、横2ドット、縦1ドット
端のブロックと壁との間のすきまは2ドット

図15



SPRITE$(0)	&B10000000=&H80	ボールのスプライト
SPRITE$(1)	&B00010000=&H10	ラケットのスプライト(極小)
SPRITE$(2)	&B00111000=&H38	〃 (小)
SPRITE$(3)	&B01111100=&H7C	〃 (中)
SPRITE$(4)	&B11111110=&HFE	〃 (大)



(注) どのスプライトにも左のような
横8×縦7の空白の部分が
上図の物の下についている

たスプライト・パターンはその結果として決めたものです．さらに実際に表示するときには，SCREEN命令によりスプライトを縦横各２倍の大きさにして用いることにします．ボールや最小のラケットで横２×縦２ドット，最大のラケットで縦２×横14ドットの大きさとなるわけです．

アプローチ２ ラケットの動かし方

1．テーブルについて

ラケットは左右方向に動かすのですから，[←]と[→]のキーを使って操作するのがよいでしょう．[←]や[→]のようなカーソルキーが押されているかどうかを調べるには，STICK関数を用います．しかし，STICK関数を使いながら，きちんと動作するようにするには（[↑]や[↓]のキーも同時に押した場合などを考えにいれると）多少の工夫を要します．ここでは[→]が押されているときには１という値，[←]が押されているときには－１という値を得ることにしましょう．すると**リスト20**のようなプログラムがいるわけです．

リスト20

```
10 A=STICK(0)
20 IF A>5 THEN I=-1:GOTO 40
30 IF A>1 AND A<5 THEN I=1 ELSE I=0
40 REM Iに あたいが゛ はいった
```

時間のかかることが問題でなければこのままでもよいのですが，ここではなるべく速くしたいのでテーブルを使います．ここでいうテーブルとは，家具のことではなく，英語で「表」という意味でのテーブルです．プログラムの手法で「テーブルを使う」というのは，配列変数などにあらかじめ何かの「対応表」を作っておいて，必要なときにすぐに取り出して使えるようにしておくことをさします．今の例で具体的に言うと，STICK関数と左（－１），右（１）の間には，**表３**のような対応関係がありますから，**リスト21**のように，あらかじめ１つの配列変数に対応関係の表を作っておけば，そのあとはいつでもほしい値が求められるわけです．

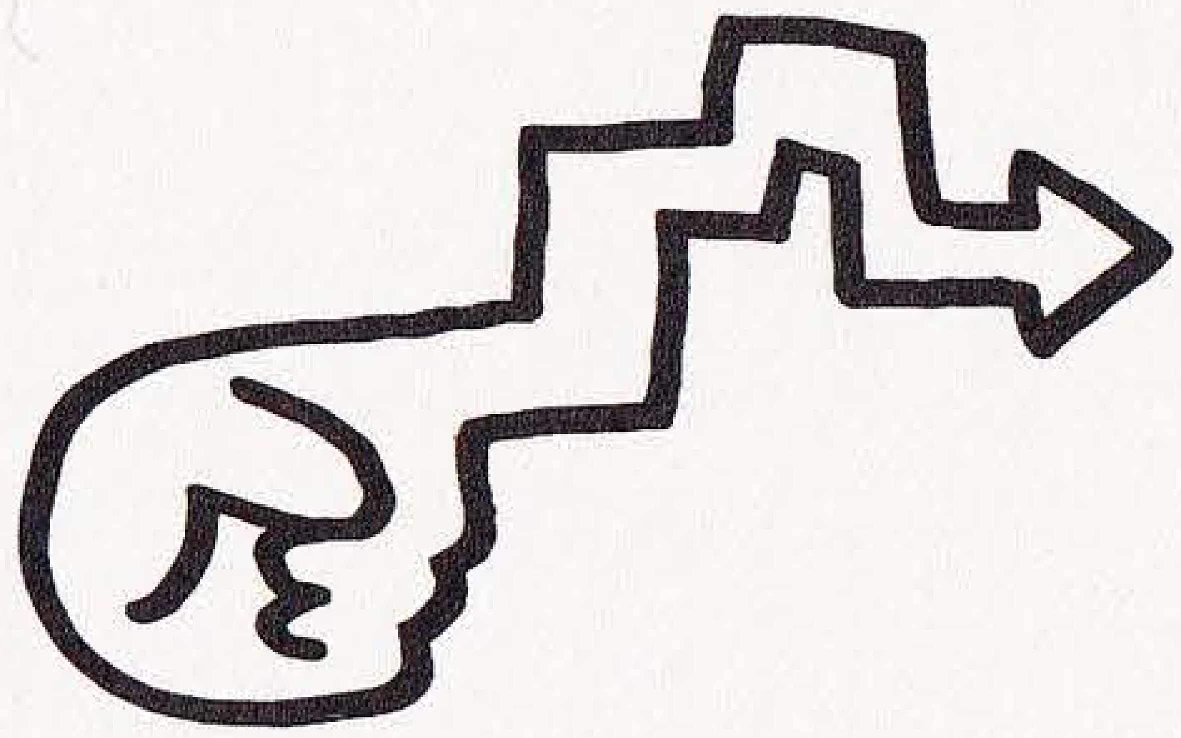
このやり方ならば，リスト20では３行かかった部分が，１行（次のリストの30行）ですんでしまいます．

リスト21

```
10 FOR J=0 TO 8:READ R(J):NEXT
20 DATA 0,0,1,1,1,0,-1,-1,-1
30 I=R(STICK(0))
```

表3

STICK関数の値	欲しい値
0	0
1	0
2	1
3	1
4	1
5	0
6	−1
7	−1
8	−1



2．ラケットの動かし方

さて，上のような配列Rを作っておいたと仮定して，ラケットを動かすにはどうしたらよいでしょうか．ラケットの横座標をRXとすれば，毎回，

RX＝RX＋R(STICK(0))

あるいは

RX＝RX＋5＊R(STICK(0))

とすることがまず考えられます．しかし実際にこれでやってみたところ，イマイチ動きがしっくりきませんでした．解決法として，タイマー割り込みを使って，頻繁にラケットを動かせるようにすることも考えましたが，全体的なスピードが落ちることが予想され，やめることにしました．結局採用したのは，次の**リスト22**にあげる方法です．

660行は，Gが4以下であればGは1増え，Gが5であれば――(G＝5)は正しいので，その論理値は－1になり，結局，

G＝G＋1－1

リスト22

```
640 I=R(STICK(0))
650 IF I=0 THEN G=0:GOTO 700
660 G=G+1+(G=5)
670 RX=RX+I*G
680 IF RX<18 THEN RX=18
690 IF RX>162 THEN RX=162
700 REM RXが もとまる
```

となり，もとの5のままというわけです．

このような論理式を使った文というのは見なれていない人も多いでしょう．論理式は，その式の値が正しければ－1を値にとり，まちがっていると0を値にとります．ですから，660行のG=G＋1＋(G＝5)でいうと，G＝5が正しくないとき，つまり5以外のときはG=G＋1と同じこととなり，G＝5の場合だけが，G=G＋0となるわけです．

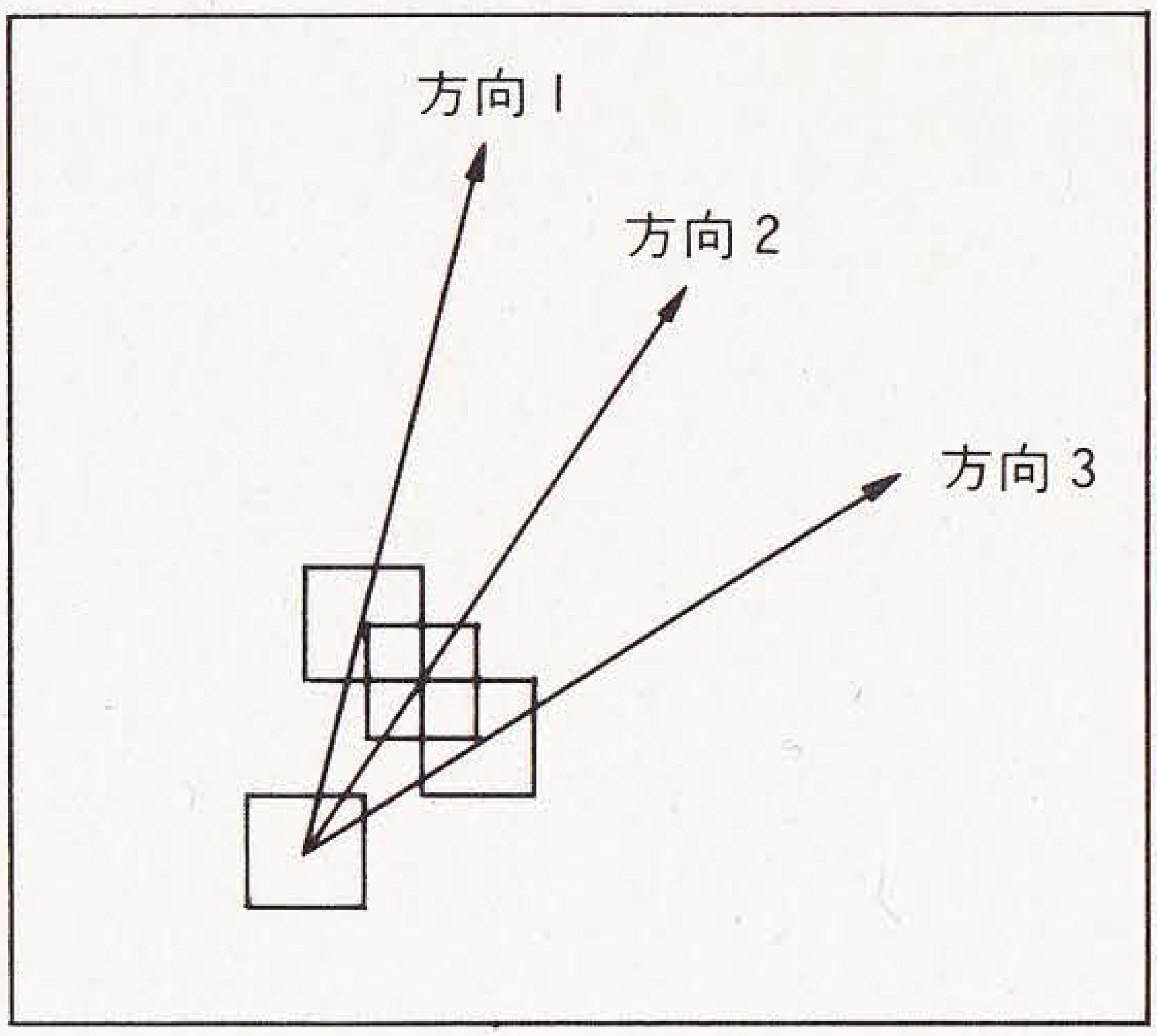
話をリスト22のプログラムに戻します．キーを押してなかった状態から，たとえば[→]のキーを押しはじめるとどうなるでしょう．リストをよく読んでみればわかると思いますが，左方向への速度の増え方は，最初このルーチンに来たときは1だったのが，2，3，4，5と増え，5以上は増えないということがわかります．したがって，このルーチンを使えば細かな動きもさせられるし，押し続けていればスピーディーな動きもさせられるわけです．

アプローチ3 ボールの動かし方

3章の南極米・ソ戦のICBMの方向計算では，方向を無段階的に変えられるようにしたのですが，ここではいくつかの方向だけに限定してしまいます．その理由は，計算を楽にすることが1つと，もう1つはオリジナルのブロックくずしでもやはりその方法を採っているからです．

図16を見てください．これは，ボールが右上方向に進んでいるときの1回あとの位置を示したものですが，図にあるような3通りの方向に制限してしまいます．実際に進む方向は，これに加えての左右が逆なもの，上下が逆なものがあり，計12パターンとなります．

図16



一回に進む横方向ドット数をU，縦方向のドット数をVとすると，図の方向1の場合であれば(U，V)＝(1，4)，2の場合なら(2，3)，3の場合なら(3，2)となります．この3つの場合をそれぞれ，0，1，2の数字で区別して，

U(0)＝1：V(0)＝4
U(1)＝2：V(1)＝3
U(2)＝3：V(2)＝2

と配列に置いておくことにします．さらにこれだけでは，12方向のうち3方向しか表現できませんから，左に行くか右に行くか，上へ行くか下に行くかを区別する2つの変数MXとMYを導入します．

左→MX＝1，右→MX＝−1
上→MY＝−1，下→MY＝1

と定めておけば，1回に進む座標の数は，

(MX＊U(D)，MY＊V(D))

と表わされます．ただしこのDは0か1か2の値をとる変数とします．

なぜ，わざわざこんなややっこしいことをするのか，と思う人もいるかもしれません．しかし，あとで全体のプログラムを見ればわかるように，この方法によってボールの進む向きの変更が楽になるのです(他にもやり方はいろいろありますが)．

結局ボールの動かし方としては，ボール位置を(BX，BY)としたときに，

BX＝BX＋MX＊U(D)：BY＝BY＋MY＊V(D)

という計算を毎回行なえばよいことになります．

アプローチ4 ボールの衝突の判定法

壁とボール，ブロックとボール，ラケットとボール，と調べるべきことはたくさんあります．多くのことを調べ，かつ実行速度を速いものにするためにはいろいろと工夫をしなければなりません．おそらくいちばん速い方法は，巨大なテーブルを作っておくことなのでしょうが，あまり現実的ではありません．そこでここでは画面の色を読んで判断することにしました．

ブロックや壁にはいろいろな色がついています．それに対して地の部分は透明ですから，ボールの位置の画面が透明ならば，ブロックや壁との衝突は調べなくてもすむことになります．さらに進んで，各段のブロックの色や壁の色がそれぞれ異なれば，どの色かを知るだけで，何段目のブロックに当たったかがわかるわけです．そこで，次のようなテーブルを作って(**表4**)，配列変数Cに入れておくことにしました．こうすると，たとえばある場所の色コードがEであるとすると，C(E)の値を0，1～6，7の3通りに分けるだけですむことになります(もとの色コードのままでは場合分けがたいへんです)．

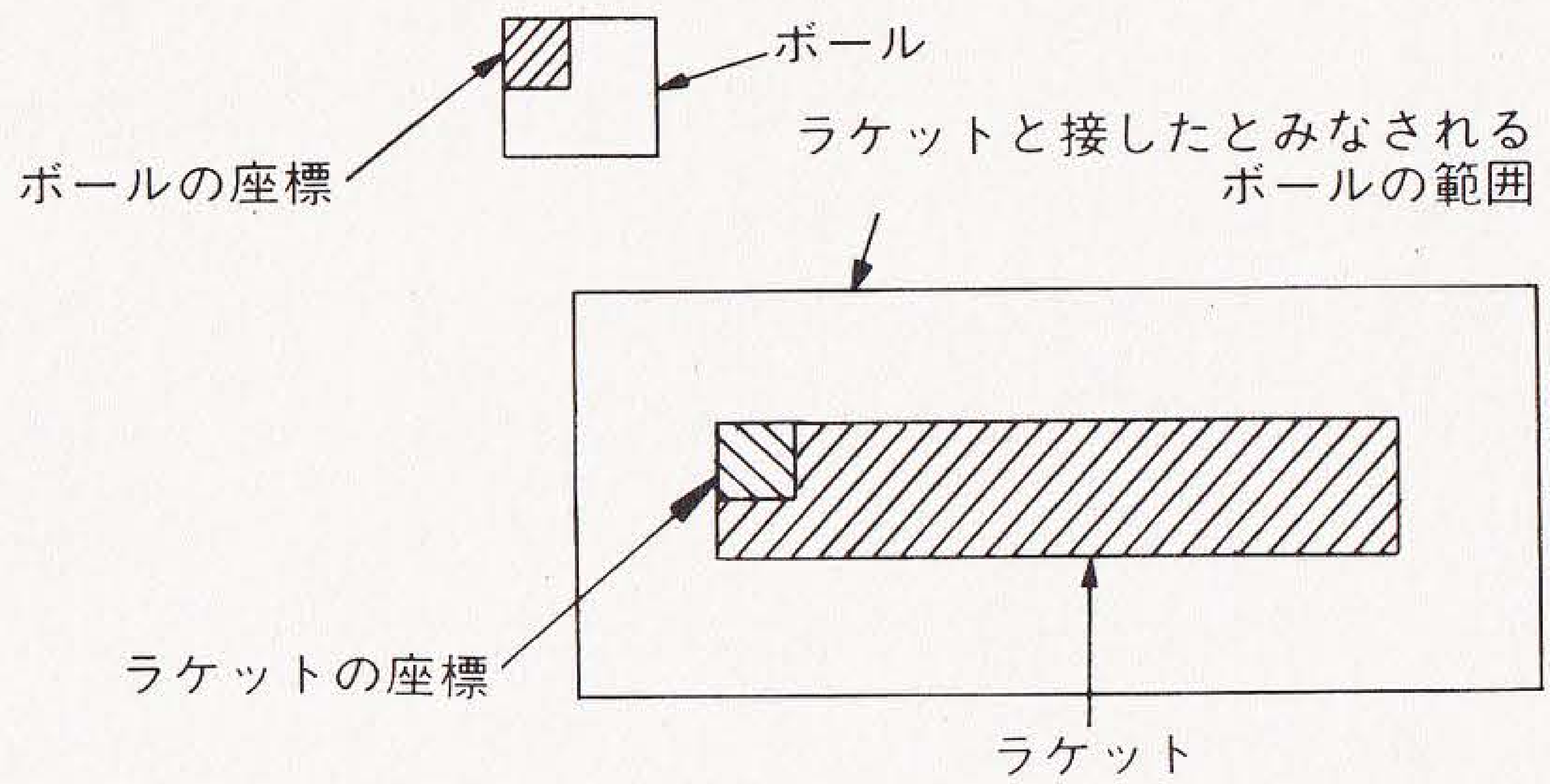
表4

色	色コード	C(色コード)	意味
明るい黄	11	0	壁ということを示す
暗い赤	6	1	上から1～6段目のブロックだということを示す
赤	8	2	
明るい赤	9	3	
黄	10	4	
明るい緑	3	5	
緑	2	6	
その他の色	0など	7	地の部分であることを示す

最後に，ラケットとの衝突とボールが画面のいちばん下までいったときの判定

を考えます．これは色コードでは判断できないので，別にする必要があります．画面のいちばん下にいったかどうかは，ボールの縦座標を調べることにより簡単にできます．ラケットとの接触を調べるのはこれに比べてめんどうですが，ボールとラケットの座標の差を調べればよいでしょう．ボールとラケットが重ならなくても接っすればよいのですから，**図17**のようにボールの位置を広めにとっておきます．また，PUT SPRITE文で使った座標は，スプライトの左隅の座標であって，ラケットの中央の座標でないことに注意する必要があります．

図17

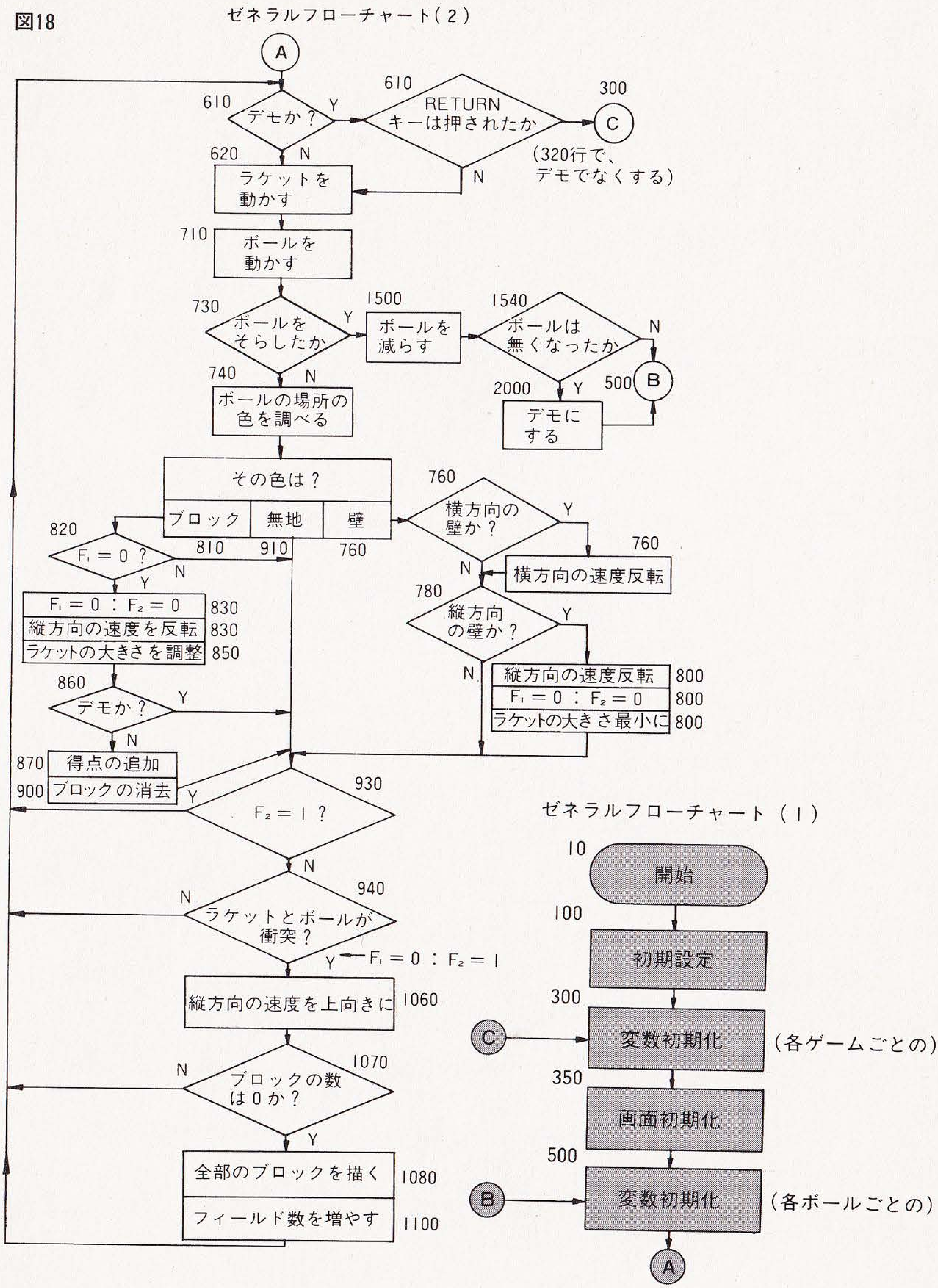


アプローチ5 全体の流れ図を書く

図18に示したのがブロックくずしのゼネラルフローチャートです．ここでF_1，F_2，デモといった言葉がいきなり使われているので，これらについて説明しましょう．

F_1は，ブロックとボールとの衝突があったかどうかを示す，「フラグ」と呼ばれるものです．次の図を見てください(**図19**)．左下のほうからきたボールの動きとして①をとるか②をとるか――．①ではあまりに簡単にブロックを消せることになってしまうので，②のほうをとることにします．①の動きなら，「ブロックにあたったら上下方向の速度を反転させる」という論理ですべてを表わせるわけですが，②では，(a)のブロックに当たったあとにぶつかる(b)のブロックとの接触は，無視しなければなりません．それをどう解決するか――．「ボールの動く向きが上から下へ行くときは無視すればよいのではないか」と思う人もいるかもしれません．しかし次の図の場合のように(**図20**)，上の壁にあたってから下のほうに

図18


ゼネラルフローチャート（2）
A
610
デモか？
Y
N
610
RETURN
キーは押されたか
300
C
（320行で、
デモでなくする）
N
620
ラケットを
動かす
710
ボールを
動かす
730
ボールを
そらしたか
Y
1500
ボールを
減らす
1540
ボールは
無くなったか
N
500
B
2000
Y
デモに
する
740
N
ボールの場所の
色を調べる
その色は？
ブロック
無地
壁
820
810
910
760
$F_1=0$？
N
Y
$F_1=0$：$F_2=0$ 830
縦方向の速度を反転 830
ラケットの大きさを調整 850
860
デモか？
Y
N
870 得点の追加
900 ブロックの消去
760
横方向の
壁か？
Y
760
横方向の速度反転
N
780
縦方向
の壁か？
Y
N
縦方向の速度反転 800
$F_1=0$：$F_2=0$ 800
ラケットの大きさ最小に 800
930
Y
$F_2=1$？
N
940
N
ラケットとボールが
衝突？
Y
$F_1=0$：$F_2=1$
縦方向の速度を上向きに 1060
1070
N
ブロックの数
は0か？
Y
全部のブロックを描く 1080
フィールド数を増やす 1100
ゼネラルフローチャート（1）
10
開始
100
初期設定
300
C
変数初期化
（各ゲームごとの）
350
画面初期化
500
B
変数初期化
（各ボールごとの）
A

動いてきたボールはしっかりと衝突してくれなければ困るのです.

図19

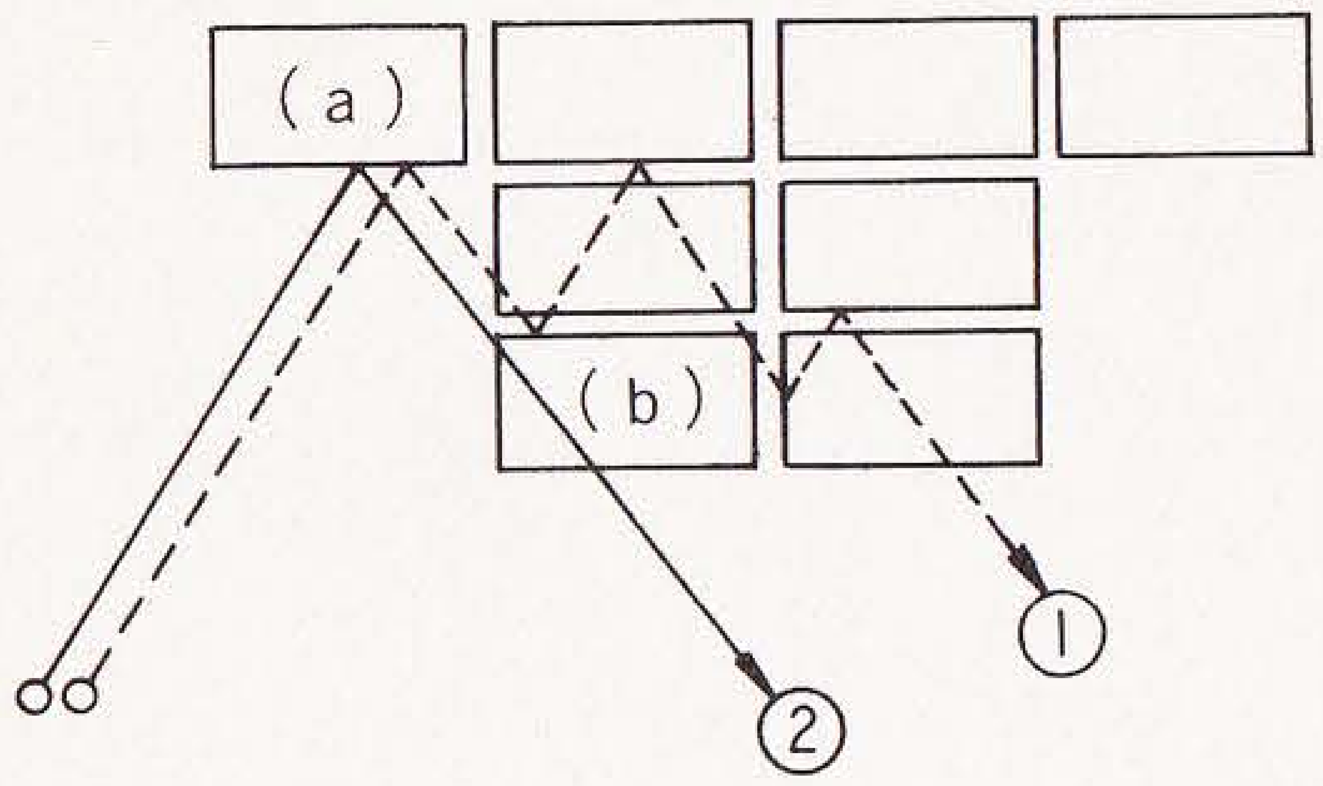
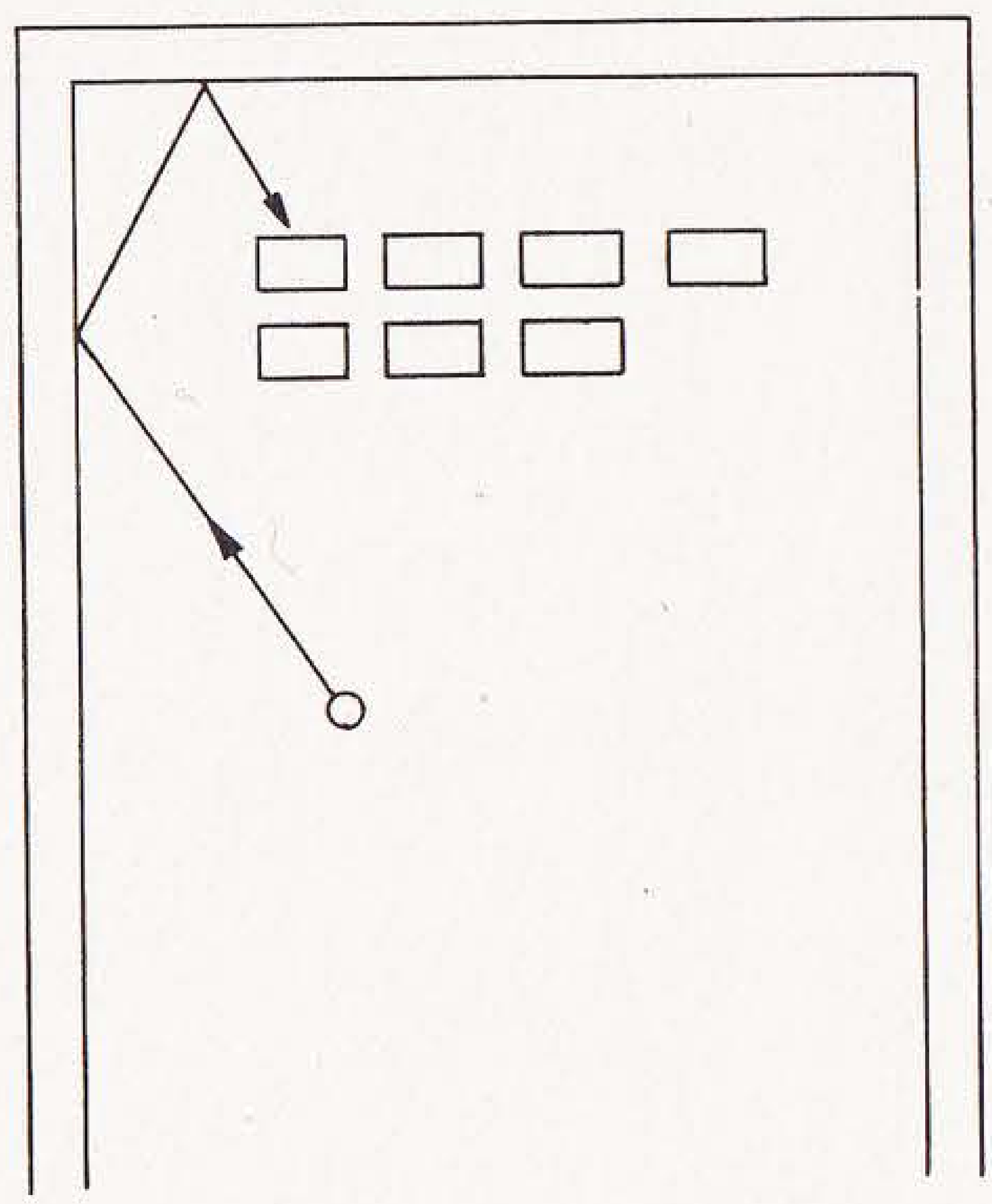


図20



そこでフラグというものを用意し，ブロックと衝突を起こしてよいかどうかを，フラグの状態により決めるという方法がとられるのです.

①ブロックにぶつかったらフラグをオンにし，それ以降ボールがブロックにぶつかっても衝突は起こらないものとする.

②ラケットあるいは，上の壁にぶつかったらフラグをオフにし，それ以降にボールがブロックに当たったら衝突を起こさせる.

上のように決めておけば，問題は解決されます．ゼネラルフローチャートでは，F_1をオンにすることを，$F_1=1$，オフにすることを$F_1=0$と書いてあります.

同様にF_2は，ラケットとボールの衝突があったかどうかを示すフラグです．このフラグは，ボールがラケットに連続して当たったように解釈されることがないように，と作られたフラグです.

最後にデモ(デモンストレーション)ですが，この本のブロックくずしのプログラムを自分の機械に打ち込んで走らせてみればわかるように，ゲーム終了後に，コンピューターが自動的にラケットを動かすデモが行なわれます．プレー中とデモのときとで異なるのは，このラケットの動きの他に,

①デモのときには得点ははいらない

②デモのときにはボールがぶつかってもブロックは消えない

といったことがあります．しかし，デモのときもプレー中も同じルーチンを使用するため，そのルーチンの中には，ところどころにデモかどうかによって処理の仕方を変える部分が出てくるわけです．

3節 プログラムを作る

3章のゲームまでは多少細かく解説してきましたが，この章では変数リストとプログラムリストにつけた解説の他に，2，3のポイントについて述べるだけにとどめます．その他は皆さんの課題というだけにしておきましょう．ここまで読んできた皆さんなら，変数リストとプログラムリスト，そして前節に掲げたゼネラルフローチャートを見て，さらに知らない命令については，BASIC文法書を開いて考えれば，プログラムの内容はおのずとわかってくることでしょう．

アプローチ1 変数リスト

B———残りのボールの数：初期値は4

N———残りのブロックの数：初期値は54（9個×6列）

S———得点（SCORE）

HS———最高得点（HIGH SCORE）

FD———フィールド数（ブロックを全部消すごとに数が増える）

DM———デモ実行中なら1，実行中でなければ0を値に持つ．

W———ラケットの大きさを示す．W＝1——横2ドット

W＝2——横6ドット

W＝3——横10ドット

W＝4——横14ドット

RX———ラケットの位置の横座標

RY———ラケットの位置の縦座標

BX———ボールの横座標

BY———ボールの縦座標

MX———ボールの横方向の進む向き（右のとき1，左のとき－1）

MY———ボールの縦方向の進む向き（上のとき－1，下のとき1）

G————1回に動くラケットの距離（ドット単位）

D————ボールの進む角度を示す．0，1，2のうちどれかの値をとる．

F————F_1フラグ．ブロックとボールとの衝突を無視するときには1，しないときには0を値にとる．

F2———F_2フラグ．ボールとラケットとの接触を無視するときには1，しないときには0を値にとる．

S(i)——上からi段目のブロックを1個消したときに得られる得点を示す．

C(i)——カラーコードiの色についての情報を持つ．値が1～6なら上から数えてそれぞれの段のブロックの色であることを示す．値が0なら壁の色，7なら地の色であることを示す．

R(i)——STICK関数の値がiのときに，それが左方向に押されたものなら1を，右方向に押されたものなら－1を値に持つ．

U(i)——i番の角度（iは0～2；変数Dで示される値）を持つボールが，1回の移動で動く横方向の距離（ドット単位）

V(i)——i番の角度を持つボールが1回の移動で動く縦方向の距離（ドット単位）．

H(i)——上からi段目のブロックを消したときに変数Wがとる値．ただし消したときにすでにWの値がH(i)以下ならWは変わらない．

アプローチ2 プログラム略説

88ページの**リスト23**がブロックくずしの全プログラムです．この中で，ちょっと難しそうなところや，説明を加えたほうがよいところを選んで解説を加えます．

(1)115～135行

```
115 FOR I=1 TO 6:READ S(I):NEXT
120 FOR I=0 TO 15:READ C(I):NEXT
125 FOR I=0 TO 8:READ R(I):NEXT
```

```
130 FOR I=0 TO 2:READ U(I),V(I):NEXT
135 FOR I=1 TO 6:READ H(I):NEXT
```

ここはデータをどんどん読み込んで，配列変数に代入している部分です．データは8000行以降にあります．

(2)150～170行

```
150 SPRITE$(0)=CHR$(&H80)+STRING$(7,0)
155 SPRITE$(1)=CHR$(&H10)+STRING$(7,0)
160 SPRITE$(2)=CHR$(&H38)+STRING$(7,0)
165 SPRITE$(3)=CHR$(&H7C)+STRING$(7,0)
170 SPRITE$(4)=CHR$(&HFE)+STRING$(7,0)
```

スプライトを定義している部分ですが，各スプライトとも先頭の１文字＋７文字のCHR$(0)をつなげているのと同じだということを理解してください．

(3)175行

```
175 I=RND(-TIME)
```

TIMEというシステム変数は時間ごとにちがった値をとるので，乱数系列を決めるときはよくこの文を使います．これを行なわないと，プログラムをRUNさせるたびにいつも同じ乱数が発生することになり，不都合なのです．

(4)180～205行

```
180 SOUND 0,0:SOUND 1,8
185 SOUND 2,0:SOUND 3,2
190 SOUND 4,0:SOUND 5,4
195 SOUND 7,56:SOUND 8,16
200 SOUND 9,16:SOUND 10,16
205 SOUND 11,0:SOUND 12,16
```

音を作り出すPSGのレジスタの初期設定を行なっています．180～190行では音の高さを設定しています．195～205行ではエンベロープを用いた音を発生させることを表わしています．これらの命令を実行した上で，

SOUND　13,　0

を実行すると，ビーンという音が発生します．

(5)610行

```
610 IF DM AND A$=CHR$(13) THEN 300
```

前にも述べましたが，IF，変数　THEN……の形でプログラムがあったら，その変数が0でないかぎりTHEN以降は実行されます．つまり，

IF　変数　＞＜0　THEN……

と同じです．この文の場合も，

610 IF DM＞＜0 AND A$＝CHR$(13) THEN…

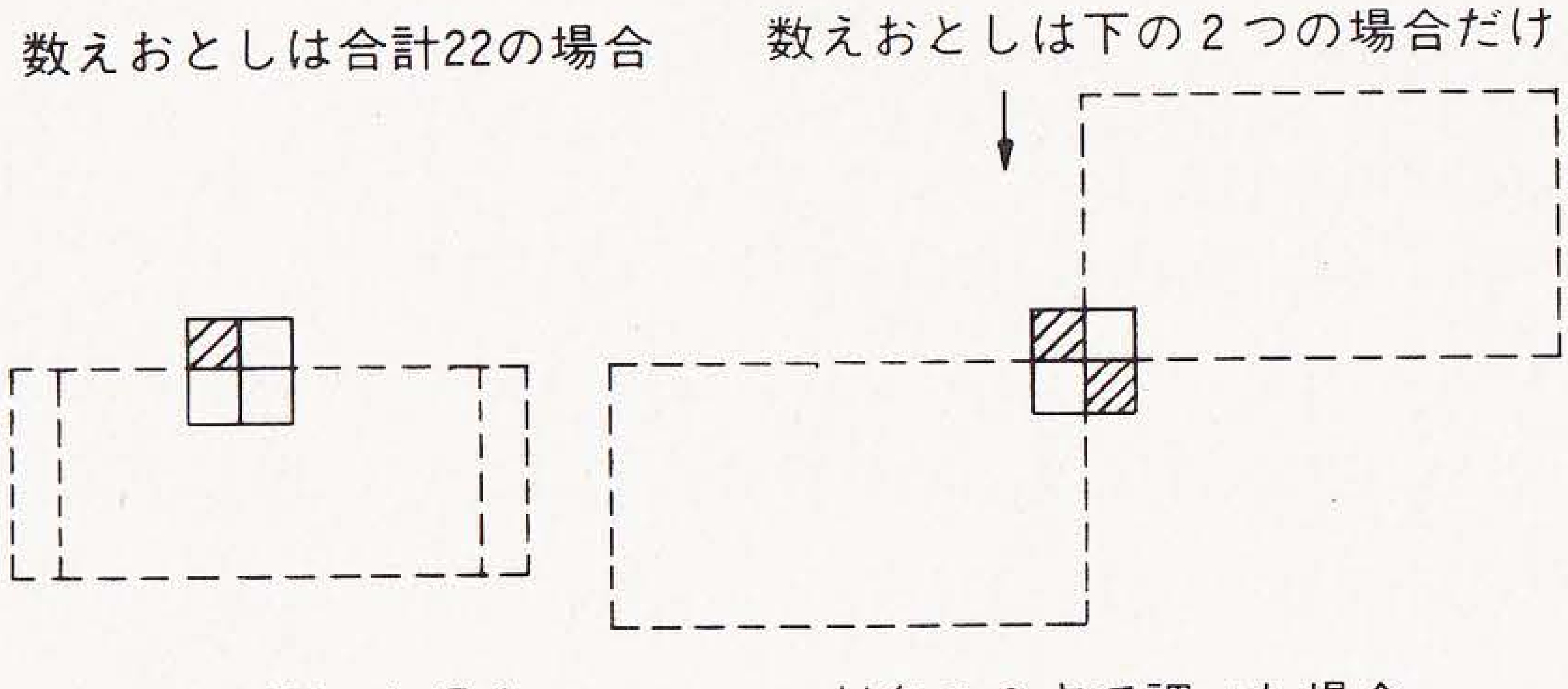
と書けます．

(6)740行

```
740 C=C(POINT(BX,BY) OR POINT(BX+1,BY+1))
```

POINT(BX, BY)でボールの位置の色が調べられるわけですが，ボールは1点ではなく，縦2×横2ドットから成り立っています．したがって，厳密にはこの4点すべてを調べなければボールの場所の色はいえないわけですが，実際には対角の2点を調べれば，ほとんどもれなく調べることができます．**図21**を見てください．1点だけで1つのブロックにぶつかったかどうかを調べると，22通りの数

図21



え落としがある(ブロックは14×7ドットでできているので)のに対し，対角の2点で調べれば，数え落としは2通りだけですむのです．4点全部を調べると時間

がかかるので，このあたりが現実的というわけです．また，

(POINT (BX, BY) OR POINT (BX＋1，BY＋1))

というようにORで結んでいますが，(BX, BY)と(BX＋1, BY＋1)の2点の色の間には，

①同じ色
②片方だけ透明(カラーコード0)
③両方とも透明

のどれかの関係が成り立っています．これはレイアウトの時点で，透明以外の2色が接することがないように配慮したおかげです．もしもこの条件が成り立っていないと，たとえば，

(2 OR 4)＝6

などという関係が成り立つので，2点の色のORをとることにより，全く違う色をあらわすことがあります．

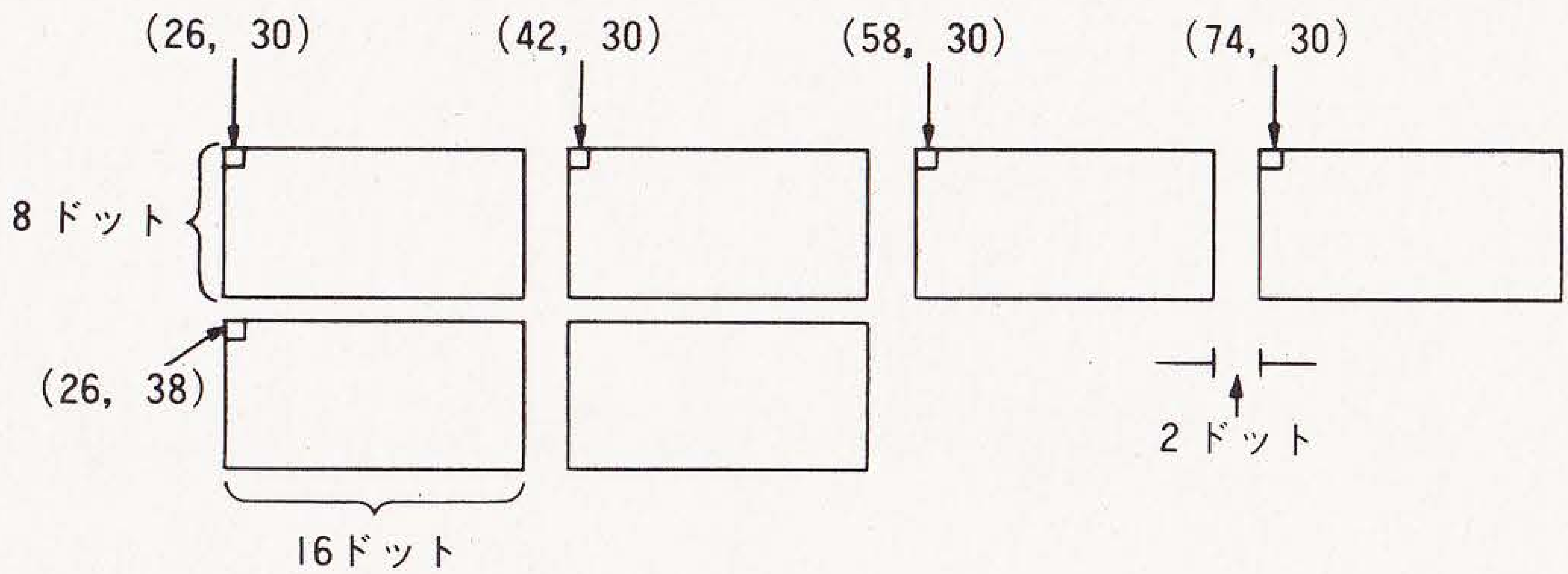
(7)890行

```
890 X=((BX+7) AND &HF0)-6
```

ここでも論理演算が出てきました．

これはブロックの左肩の横座標を求めるためのものです．**図22**を見てください．この図のようにこのブロックくずしでは，(26，30)の位置を左上の肩とするブロックを画面のいちばん左上に置いて，横には16ドットごとに，縦には8ドットごとにブロックがあります．

図22



そして，890行の計算を行なうと，BXの値が25から40ならXは26になり，また41から56ならXは42になります．このようにブロックに当たったボールの位置がわかれば，そのブロックの左端のX座標が求まるわけです．また，この890行は，割り算，かけ算を用いても表わすことができます．その場合は，

```
890 X2=(BX+7)¥16:X=X2*16-6
```

のようになります．

ブロックくずし全リスト

リスト23

```
10 REM   ブロック クズシ
20 REM
30 REM    Ver 1.60
40 REM
100 REM しょきせってい ルーチン
105 DEFINT A-Z
110 DIM S(6),C(15),R(8),U(2),V(2),H(6)
115 FOR I=1 TO 6:READ S(I):NEXT
120 FOR I=0 TO 15:READ C(I):NEXT
125 FOR I=0 TO 8:READ R(I):NEXT
130 FOR I=0 TO 2:READ U(I),V(I):NEXT
135 FOR I=1 TO 6:READ H(I):NEXT
140 SCREEN 2,1,0
145 OPEN"GRP:" FOR OUTPUT AS #1
150 SPRITE$(0)=CHR$(&H80)+STRING$(7,0)
155 SPRITE$(1)=CHR$(&H10)+STRING$(7,0)
160 SPRITE$(2)=CHR$(&H38)+STRING$(7,0)
165 SPRITE$(3)=CHR$(&H7C)+STRING$(7,0)
170 SPRITE$(4)=CHR$(&HFE)+STRING$(7,0)
175 I=RND(-TIME)
180 SOUND 0,0:SOUND 1,8
185 SOUND 2,0:SOUND 3,2
190 SOUND 4,0:SOUND 5,4
195 SOUND 7,56:SOUND 8,16
200 SOUND 9,16:SOUND 10,16
205 SOUND 11,0:SOUND 12,16
300 REM ゲーム の はじまり
310 B=4:N=54:S=0
320 FD=1:DM=0
```

115〜130行：配列変数の初期設定

150〜155行：←ボール用スプライトの設定

160〜170行：ラケット用スプライトの設定

175行：←乱数系列の初期化

180〜200行：PSGレジスタの初期設定

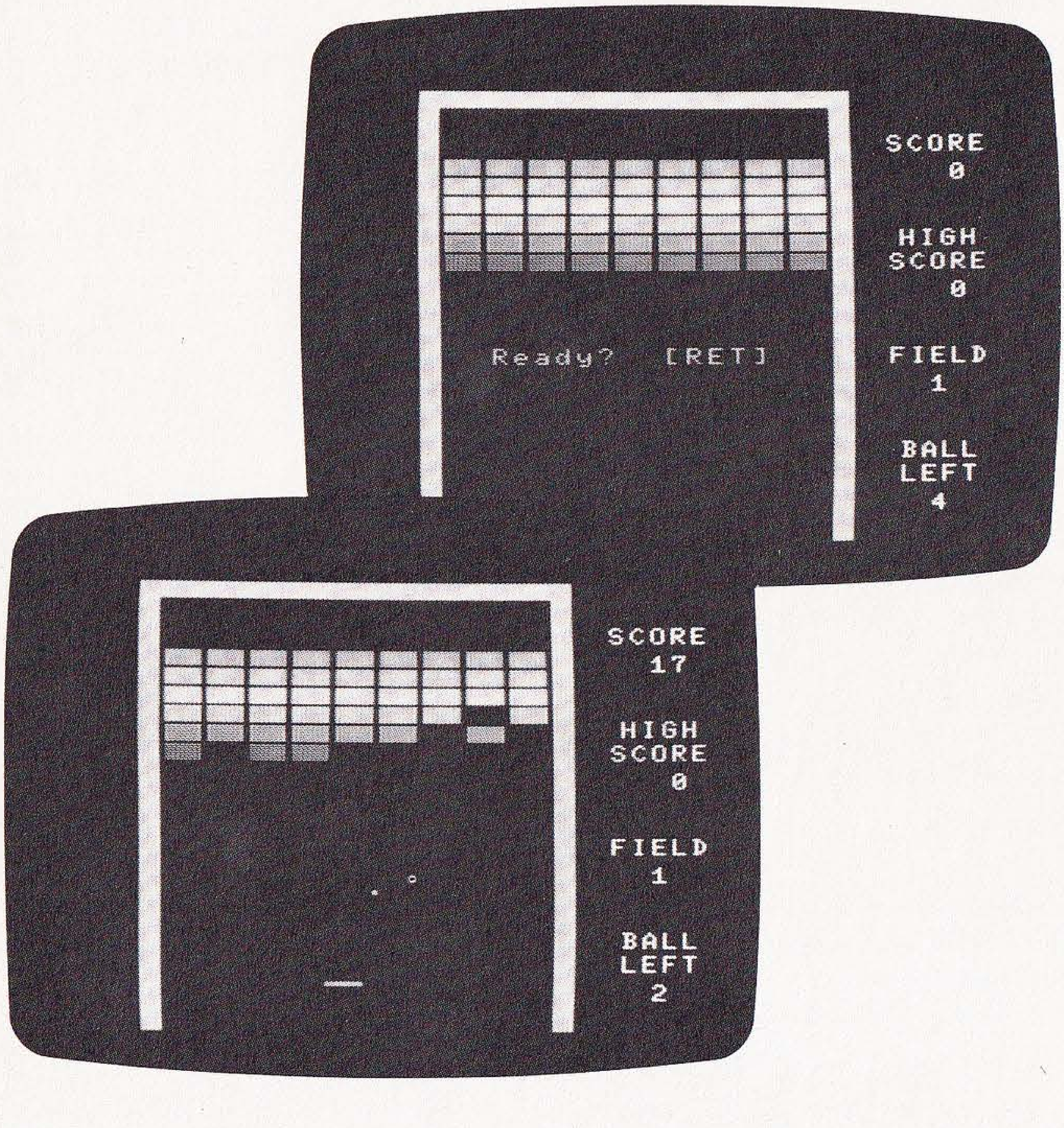
```
330 REM ガ゛メン しょきせってい ルーチン
340 PUTSPRITE0,(-8,-8),,0     } スプライトを消去
350 PUTSPRITE1,(-8,-8),,W     }
360 COLOR 11,0,1:CLS
370 LINE(16,0)-(23,191),,BF   } 壁を描く
380 LINE(16,0)-(169,7),,BF    }
390 LINE(170,0)-(177,191),,BF
400 COLOR 14
410 GOSUB 4000
420 GOSUB 3000   }
430 GOSUB 5000   }
440 GOSUB 5100   } 画面に種々の文字を描く
450 GOSUB 5200   }
460 GOSUB 5300
470 REM メッセージ゛を だ゛す
480 GOSUB 5500
500 REM へんすう しょきせってい
510 W=4:F2=0
520 RX=96:RY=170
530 BX=RND(1)*135+24:BY=80    }
540 MX=(RND(1)>.5)*2+1:MY=1   } ボールの方向, 場所の初期設定
550 D=RND(1)*3                }
570 REM
580 REM メイン ルーチン
590 REM
600 A$=INKEY$
610 IF DM AND A$=CHR$(13) THEN 300 ←デモ中にRETが押されたら300へ
620 REM ラケット を うご゛かす ルーチン
630 IF DM THEN RX=BX+U(D)*MX-6:GOTO 680 ←デモ中ラケットの動き
640 I=R(STICK(0))
650 IF I=0 THEN G=0:GOTO 700  }
660 G=G+1+(G=5)               }
670 RX=RX+I*G                 } ラケットを動かす
680 IF RX<18 THEN RX=18       }
690 IF RX>162 THEN RX=162     }
700 PUT SPRITE1,(RX,RY),11,W
710 REM ボ゛ールを うご゛かす ルーチン
720 BX=BX+U(D)*MX:BY=BY+V(D)*MY
730 IF BY>190 THEN 1500 ←ボールを逃がしたら1500行へ
740 C=C(POINT(BX,BY)OR POINT(BX+1,BY+1))
750 IF C=7 THEN 910 ELSE IF C THEN 810
760 IF BX<25THEN MX=1:SOUND13,0 ←左の壁との衝突判定
770 IF BX>167THEN MX=-1:SOUND13,0 ←右の壁との衝突判定
780 IF BY>8 THEN 910 ←上の壁との衝突判定
```

```
790 SOUND 13,0
800 MY=1:F=0:F2=0:W=1 ←ラケットの大きさを最小に
805 GOTO 910
810 REM ﾌﾞﾛｯｸ に しょうとつ
820 IF F THEN 910
830 F=1:F2=0:MY=-MY
840 SOUND 13,0
850 IF W>H(C) THEN W=H(C) ←ラケットの大きさの調整
860 IF DM THEN 910
870 Y=22+8*C:S=S+S(C):N=N-1 ←ブロックの縦座標の計算
880 GOSUB 5000 ←スコアの表示
890 X=((BX+7) AND &HF0)-6 ←ブロックの横座標の計算
900 LINE(X,Y)-(X+13,Y+6),0,BF ←ブロックを消す
910 REM
920 PUT SPRITE0,(BX,BY),15,0 ←ボールの表示
930 IF F2 THEN 1120 ←1度ラケットに当たっているなら1120へ
940 REM ﾎﾞｰﾙ と ﾗｹｯﾄ の しょうとつ を しらべる
950 X=BX-RX-6:Y=BY-RY ←相対位置の計算
960 IF ABS(Y)>2 THEN 1120
970 I=ABS(X)-W-W
980 ON SGN(I)+1 GOTO 1020,600
990 IF I=-1 THEN 1030
1000 REM ふつうに あたった とき
1010 GOTO 1040
1020 REM かどに あたった とき
1030 MX=SGN(X-.5) ←MXは1か－1になる
1040 IF RND(1)<.3 THEN D=RND(1)*3 ←ときおりボールの進む
1050 SOUND 13,0                        角度を変える
1060 MY=-1:F=0:F2=1
1070 IF N THEN 600
1080 GOSUB 3000 ┐
1090 N=54       │
1100 FD=FD+1    ├ 全部をブロックを消した像の処置
1110 GOSUB 5200 ┘
1120 GOTO 600
1500 REM ﾎﾞｰﾙ を ミスした ときの しょり
1510 PUT SPRITE0,(-8,-8),,0 ←ボールを消去
1520 B=B-1      ┐
1530 GOSUB 5300 ┘ ボールの数を1つ減らし，表示
1540 IF B=0 THEN 2000
1550 GOSUB 5500
1560 GOTO 500
2000 REM ｹﾞｰﾑ しゅうりょう しょり
2010 IF S<HS THEN 2030 ←ハイスコアか判定
2020 HS=S:GOSUB 5100 ←ハイスコアの表示
```

```
2030 COLOR 14
2040 DRAW"BM56,108":PRINT#1,"Try Again!
2050 DRAW"BM76,120":PRINT#1,"[RET]"
2060 DM=1
2070 GOTO 500
3000 REM ぜんぶの ブロックを かく ルーチン
3010 LINE(26,30)-(167,36),6,BF
3020 LINE(26,38)-(167,44),8,BF
3030 LINE(26,46)-(167,52),9,BF
3040 LINE(26,54)-(167,60),10,BF
3050 LINE(26,62)-(167,68),3,BF
3060 LINE(26,70)-(167,76),2,BF
3070 FOR I=40 TO 152 STEP 16
3080 LINE(I,30)-(I+1,76),0,B
3090 NEXT
3100 RETURN
4000 REM もじを かく ルーチン
4010 DRAW"BM192,20":PRINT#1,"SCORE"
4020 DRAW"BM196,60":PRINT#1,"HIGH"
4030 DRAW"BM192,70":PRINT#1,"SCORE"
4040 DRAW"BM192,110":PRINT#1,"FIELD"
4050 DRAW"BM196,150":PRINT#1,"BALL"
4060 DRAW"BM196,160":PRINT#1,"LEFT"
4070 RETURN
5000 REM SCORE を だす ルーチン
5010 LINE(223,39)-(192,32),0,BF
5020 PRINT#1,USING"####";S
5030 RETURN
5100 REM HIGH-SCORE を だす ルーチン
5110 LINE(223,89)-(192,82),0,BF
5120 PRINT#1,USING"####";HS
5130 RETURN
5200 REM FIELD のかず を だす ルーチン
5210 LINE(215,129)-(192,122),0,BF
5220 PRINT#1,USING"###";FD
5230 RETURN
5300 REM ボール のかず を だす ルーチン
5310 LINE(215,179)-(208,172),0,BF
5320 PRINT#1,USING"#";B
5330 RETURN
5500 REM じゅんびは いいか たずねる
5510 DRAW"BM44,110"
5520 PRINT#1,"Ready?  [RET]"
5530 A$=INKEY$
5540 IF A$<>CHR$(13) THEN 5530
5550 LINE(44,110)-(147,117),0,BF
```

- 2040–2050: メッセージの表示
- 3010–3060: 全部のブロックを描く
- 4010–4050: 画面に表示する文字を描く
- 5010–5020: スコアの表示
- 5110–5120: ハイスコアの表示
- 5210–5220: フィールド数の表示
- 5310–5320: 残りボール数の表示
- 5510–5540: RETURNを押すのを待つ

```
5560 RETURN
8000 REM かくた゛んの ホ゜イント の テ゛ータ
8010 DATA 20,12,8,5,3,1  ←ブロック1個あたりの得点のデータ
8020 REM いろ と た゛んすう の たいおう テ゛ータ
8030 DATA 7,7,6,5,7,7,1,7
8040 DATA 2,3,4,0,7,7,7,7  ←各色と段数の対応，0は壁の色
8050 REM STICKかんすう と さゆうの うこ゛きの たいおう
8060 DATA 0,0,1,1,1,0,-1,-1,-1
8070 REM U(n),V(n) を せってい する
8080 DATA 1,4,2,3,3,2
8090 REM ((ラケット の はは゛)+1)/2
8100 DATA 2,3,3,4,4,4
```



BASICゲーム教室

MSXポケットバンク 3

1984年4月5日　初版発行　　定価480円
1987年9月11日　第1版第4刷発行

著　者　安田吾郎

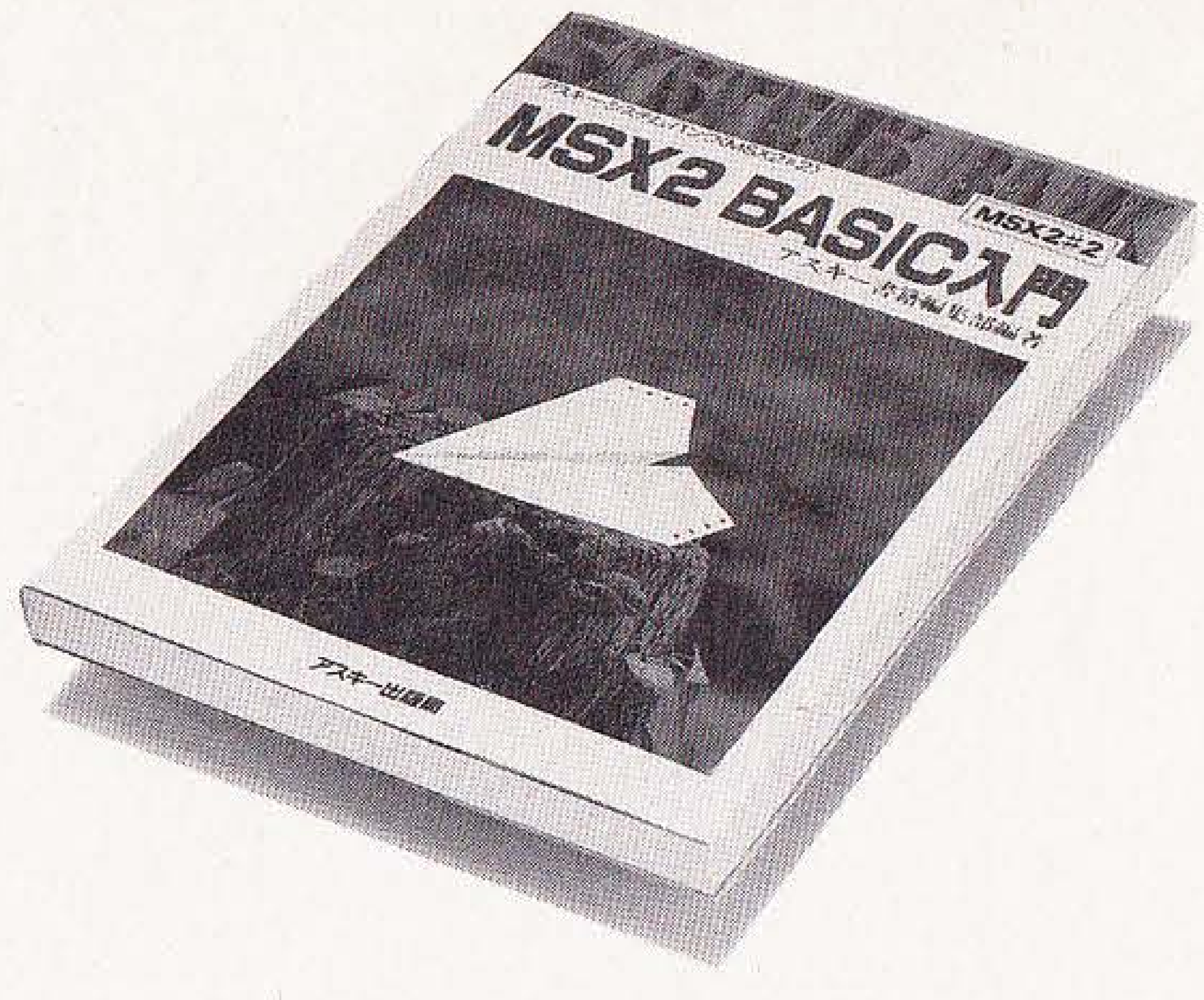
発行者　塚本慶一郎
発行所　株式会社アスキー
　〒107　東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)486-7111(大代表)
情報TEL (03)498-0299(ダイヤルイン)
出版営業部TEL (03)486-1977(ダイヤルイン)

編集担当　八木淳一
表紙担当　郷　啓子
印刷　株式会社加藤文明社印刷所

ISBN4-87148-735-0 C0055 ¥480E

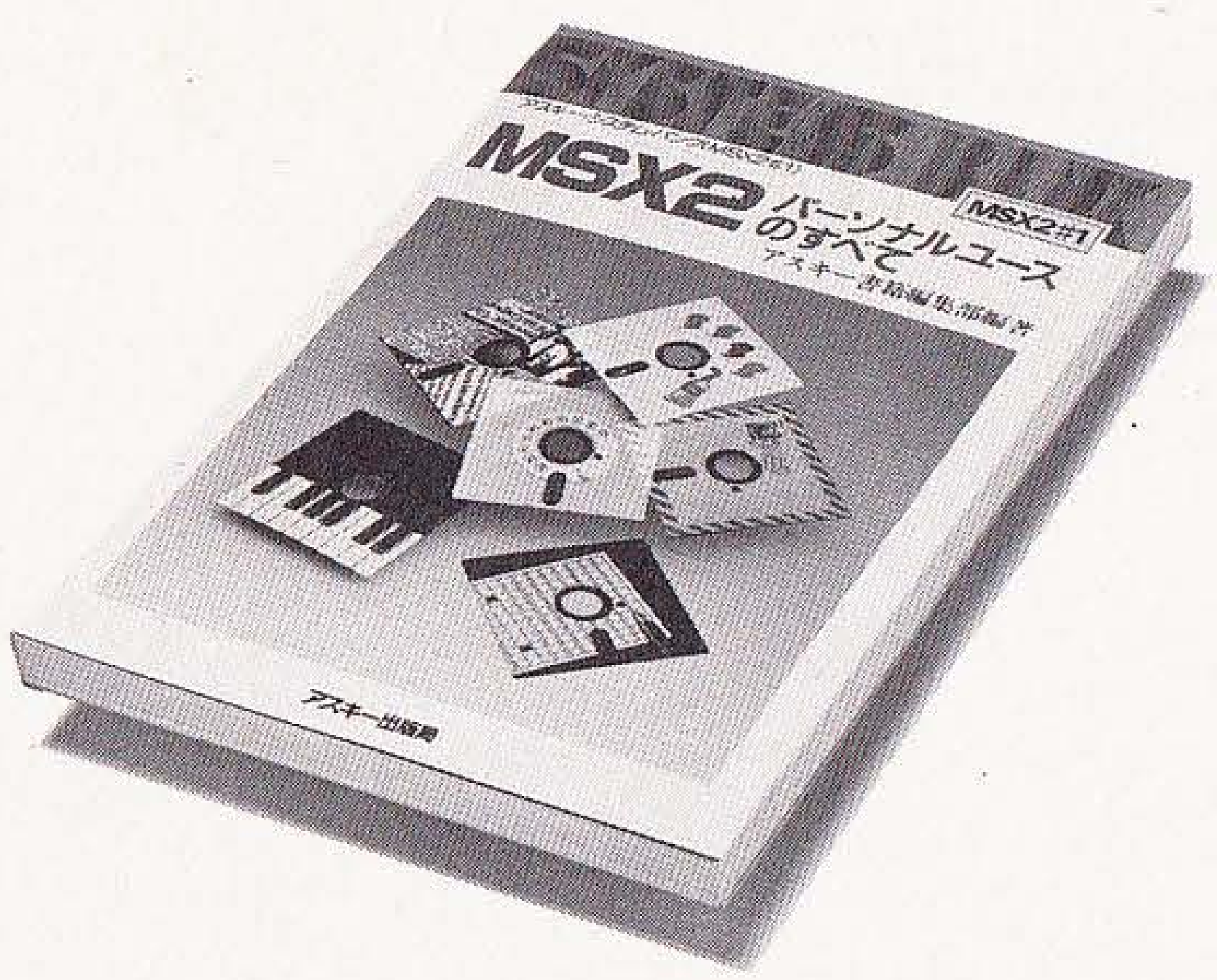
MSX2 BASIC入門

アスキー書籍編集部編著　定価2,500円（送料300円）



本格派MSX2ユーザーに贈る、MSX-BASIC(ver.2.0)入門書の決定版。文法の解説はもちろん、実際のプログラミングやデバッグで役に立つヒント、アドバイスも満載しました。平易な解説と豊富な操作例により、実際にマシンを操作しながら、BASICプログラミングを着実にマスターできます。本書を読み終えた後、どんなプログラム作りに挑戦するか――それはあなたのアイディアと熱意次第です！ MSX2独自のファイル操作やグラフィックス、画像処理などについても詳説。初心者の方に限らず、MSX2でBASICプログラムを作ろうとするすべて人に必読の一冊です。

〈内容〉
MSX2とMSX-BASIC／BASICの基礎知識／プログラミングの基本／エレガントなプログラミング／入出力装置とファイル操作／グラフィックスとビジュアル機能／プログラミングとデバッギング／APPENDIX



MSX2 パーソナルユースのすべて

アスキー書籍編集部編著　定価2,500円（送料300円）

MSX2を日常の道具として気軽に利用していくためのプログラム集。データベース、欧文ワープロ、グラフィックエディタ、通信プログラムなど、個人で利用するには十分な機能を備えた実用プログラムを満載しました。少し手を加えて自分専用のバージョンを作る楽しさもまた格別。そして、あなた自身でプログラムを作る際のヒントとなることも、本文やリストの中にきっと見つかるはずです。MSX2と、もっと親密なつきあいをしたい方にぜひお勧めします。

〈内容〉
パーソナルユース活用法／MSX2のパフォーマンス／パーソナル・ワープロ／パーソナル・データベース／パーソナル・グラフィックス／パソコン・ミュージック／パソコン・コミュニケーション

アスキー・ディスクアルバム⑲
『MSX2パーソナルユースのすべて』
本書のプログラムをディスクに収録

MSX DISK BASIC版 3.5″ 2DD
定価4,300円（送料400円）

■全国有名書店、マイコンショップでお求めください。万一品切れの場合は直接書店または弊社までお申込みください。

〒107 東京都港区南青山6－11－1 スリーエフ南青山ビル PHONE 03(486)1977　株式会社アスキー

情報BOX

アスキーブックス

MSXホーム コンピュータ読本



竹内あきら・湯浅敬・安田吾郎共著
A5変型　定価1,600円(送料300円)

〈内容〉
INTRODUCTION▷ホームコンピュータとしてのMSX／はじめてのMSX／MSXの機能／MSXの利用法／他
TUTORIAL▷BASICを知ろう／BASICをはじめよう／BASIC入門／他
REFERENCE▷変数と演算子／文字を出す／グラフィックス／サウンド／プログラム作り／実行手順の制御／他

アスキーブックス

MSXビギナーズ BASIC



児玉真之著
A5変型　定価1,500円(送料300円)

〈内容〉
はじめようMSX／BASIC基礎講座／楽しくプログラミング／グラフィック&サウンド／これからが本格派／APPENDIX

アスキーブックス

MSXグラフィック ワークブック



桜田幸嗣・蓑島聡共著
A5変型　定価1,500円(送料300円)

〈内容〉
忘れていませんかグラフィックの約束ごと、MSXの使い方／グラフィックの基本操作…覚えてほしい、あんなこと、こんなこと、／アートへのアプローチ／ゲームへのアプローチ／ランダムアプリケーション／Appendix

アスキーブックス

MSXマシン語 入門講座



湯浅敬著
A5変型　定価1,600円(送料300円)

〈内容〉
マシン語ってどんなもの／MSXのハードウェアを調べる／マシン語プログラムの作り方／メモリとレジスタのデータ転送／ロード命令の応用と増減命令／サブルーチンとBIOS／スタックとサブルーチンの利用／論理演算／マシン語の計算／Fレジスタの使い方／プログラムの制御／ビット単位のデータ操作／I／Oポートの利用／他

■全国有名書店、マイコンショップでお求めください。万一品切れの場合は直接書店または弊社までお申込みください。

〒107 東京都港区南青山6－11－1 スリーエフ南青山ビル PHONE 03(486)1977 株式会社アスキー

情報BOX

アスキーブックス
はじめて読む BASIC



高玉皓司著
A5判　定価1,200円(送料300円)

〈内容〉
ふれること慣れること〔体験編〕
知るべきことを知る〔基礎編〕
できることいろいろ〔中級編〕
表現力をつける〔グラフィック編〕
使いこなすために〔上級編〕
付録

アスキーブックス
はじめて読む マシン語



村瀬康治著
A5判　定価1,200円(送料300円)

〈内容〉
ほんとうのコンピュータと出逢うために／マシン語を学ぶ道具MCモニタ／メモリに関する基礎知識／2進法と16進法／コンピュータに直接命令できるのはマシン語だけ／CPU,メモリ,I／O,バス／ニーモニックとアセンブラ／コンピュータの中心CPU／マシン語命令の実習／やさしいプログラミング例／マシン語から広がる世界

アスキーブックス
はじめて読む アセンブラ



村瀬康治著
A5判　定価1,600円(送料300円)

〈内容〉
ソフトウェアの原点アセンブラ／アセンブラの全体像の把握／ソース・プログラムの基本的な書式／ソフトウェア開発手順とその実例／ソフトウェア開発ツールとその機能／やさしいプログラミング実習／プログラミングの基本／アセンブラの諸機能実習解説／実用プログラムの作成／デバッガ／他

アスキーブックス
はじめて読む8086



村瀬康治監修　蒲地輝尚著
A5判　定価1,600円(送料300円)

〈内容〉
マシン語から広がる世界／実行型ファイルをダンプする／実行型ファイルのメッセージを変更する／これだけは覚えて欲しいコンピュータの知識／8086CPUの基礎／マシン語命令の実習／やさしいプログラミングの実例／マクロアセンブラによるマシン語プログラミング/APPENDIX

■全国有名書店、マイコンショップでお求めください。万一品切れの場合は直接書店または弊社までお申込みください。

〒107 東京都港区南青山6－11－1 スリーエフ南青山ビル　PHONE 03(486)1977　株式会社アスキー

MSX
POCKET BANK

MSXポケットバンク3

BASICゲーム教室

CONTENTS

MSXポケットバンク・シリーズ
好評既刊 各巻480円～680円

ISBN4-87148-735-0 C0055 ¥480E

定価480円